Engineering Criminal Agents

Software Technology for Evidence-based Social Simulation

by
Ulf Lotzmann

Approved Dissertation thesis
for the partial fulfilment of the requirements for a
Doctor of Natural Sciences (Dr. rer. nat.)
Fachbereich 4: Informatik
Universität Koblenz-Landau

Chair of PhD Board: Prof. Dr. Maria A. Wimmer
Chair of PhD Commission: JProf. Dr. Kai Lawonn
Examiner and Supervisor: Prof. Dr. Klaus G. Troitzsch
Further Examiners: Prof. Dr. Maria A. Wimmer

Date of the doctoral viva: 11th January 2019

To my parents.

Acknowledgement

Many friends and colleagues have contributed considerably
— in the one or other way —
to the accomplishment of my PhD thesis.
I would like to express my sincere thanks to:

My supervisors Klaus G. Troitzsch and Maria A. Wimmer

Scott Moss,
Sabrina Scherer,
Ruth Meyer
and the colleagues of the OCOPOMO project

Martin Neumann
and the colleagues of the GLODERS project

My adviser Michael Möhring

Danke!

Abstract

This PhD thesis with the title "Engineering Criminal Agents" demonstrates the interplay of three different research fields captured in the title: In the centre are Engineering and Simulation, both set in relation with the application field of Criminology — and the social science aspect of the latter. More precisely, this work intends to show how specific agent-based simulation models can be created using common methods from software engineering.

Agent-based simulation has proven to be a valuable method for social science since decades, and the trend to increasingly complex simulation models is apparent, not at least due to advancing computational and simulation techniques. An important cause of complexity is the inclusion of 'evidence' as basis of simulation models. Evidence can be provided by various stakeholders, reflecting their different viewpoints on the topic to model.

This poses a particular burden by interrelating the two relevant perspectives on the topic of simulation: on the one hand the *user* of the simulation model who provides the requirements and is interested in the simulation results, on the other hand the *developer* of the simulation model who has to program a verified and validated formal model. In order to methodically link these two perspectives, substantial efforts in research and development are needed, where this PhD thesis aims to make a contribution.

The practical results — in terms of software — were achieved by using the multi-faceted approach mentioned above: using methods from software engineering, in order to become able to apply methods from computational social sciences, in order to gain insights into social systems, such as in the internal dynamics of criminal networks.

The PhD thesis shows the research involved to create these practical results, and gives technical details and specifications of the developed software.

The frame for research and development to achieve these results was provided mainly by two research projects: OCOPOMO and GLODERS.

Zusammenfassung

Die Dissertation mit dem Titel “Engineering Criminal Agents” demonstriert das Zusammenspiel von drei verschiedenen Forschungsbereichen, die bereits im Titel genannt sind: Im Mittelpunkt stehen Engineering (d.h. ingenieurmäßiges Vorgehen bei der Erstellung von Systemen) und Simulation, während beide Bereiche im Kontext des Anwendungsfeldes Kriminologie (und auch den damit verbundenen sozialwissenschaftlichen Aspekten) angewandt werden. Konkreter gesagt, Ziel der Arbeit ist es, zu zeigen, wie spezifische Arten von agentenbasierten Simulationsmodellen unter Berücksichtigung und Anwendung von Methoden der Softwareentwicklung erstellt werden können.

Agentenbasierte Simulation hat sich seit Jahrzehnten als nützliche Methode der Sozialwissenschaften bewährt, und ein Trend zu komplexen Simulationsmodellen ist wahrnehmbar, nicht zuletzt aufgrund der Fortschritte in Datenverarbeitungs- und Simulationstechniken. Eine bedeutende Ursache für Modellkomplexität ist die Einbeziehung von ‘Belegmaterialien’ als Grundlage von Simulationsmodellen. Solche Belegmaterialien können durch mehrere Stakeholder bereitgestellt werden und dabei deren unterschiedliche Sichtweisen auf einen Modellierungsgegenstand widerspiegeln.

Dabei ergeben sich spezifische Problemstellungen für das Zusammenspiel der beiden hier relevanten Perspektiven auf das Thema Simulation: auf der einen Seite der *Benutzer* des Simulationsmodells, der die Anforderungen liefert und an den Simulationsergebnissen interessiert ist; auf der anderen Seite der *Modellentwickler*, der ein verifiziertes und validiertes formales Modell programmieren muss. Um diese beiden Perspektiven systematisch zusammenzubringen, ist substantieller Aufwand in Forschung und Entwicklung erforderlich, wo die vorliegende Dissertation einen Beitrag zu leisten beabsichtigt.

Die in dieser Arbeit erzielten praktischen Ergebnisse — in Form von Software — wurden durch eine bereichsübergreifende Herangehensweise erreicht: Durch die Verwendung von Methoden der Softwareentwicklung konnten Methoden der computergestützten Sozialwissenschaften benutzt werden, um am Ende Einblicke in soziale Systeme — wie die internen Dynamiken von kriminellen Netzwerken — zu ermöglichen.

Die vorliegende Dissertation zeigt die Forschungstätigkeiten, die für die Erzielung der Ergebnisse herangezogen wurden, sowie liefert Details und Spezifikationen zur erstellten Software.

Rahmen für diese Forschungs- und Entwicklungstätigkeiten waren in erster Linie zwei Forschungsprojekte: OCOPOMO und GLODERS.

Contents

List of Figures

List of Tables

Chapter 1

Motivation and Problem Scope

1.1 Introduction

From the early beginnings of solving problems with help of computers, simulation has been a prominent research method. In the course of time more and more simulation techniques were developed and new application areas entered. Nowadays, for almost every real-world problem, distinct simulation approaches and techniques are available which allow to represent the characteristic and relevant aspects in order to achieve expressive and reliable results. Hence, it seems consequent that a still increasing number of scientific disciplines see simulation as an asset for the repertoire of 'classical' research methods.

In contrast to typical 'simulation-suited' disciplines such as physics or engineering, in the social sciences simulation methods gained popularity only in the 1990s, although already 30 years before e.g. Schelling (1969) unveiled the benefits of a presently very popular simulation method — based on software agents — with his famous segregation model, which is part of any (social) simulation curriculum today (Gilbert and Troitzsch, 2005).

As manifold and diverse the simulation techniques have become over time, as steady and clear the purposes of simulation have remained:

- *Training*: to allow a user of some kind of system to get used with the system, especially in rare special situations which cannot be exercised with the real system.
- *Forecast*: to attempt to predict possible developments of a system in the future.
- *Explanation*: to understand the behaviour of a system for which certain parameters are unknown due to covert system internals or due to a too high system complexity.

In the area of science, simulation can serve various objectives, not necessarily restricted to fundamental research, but not less for practical usage. One of those fields that are in the process of encountering new uses for the simulation

method is criminology. The predominant aim in this field has been to utilise simulation to predict potential 'hot spots' of crime, in order to inform police strategies and operations with this additional source of information. A more recent approach in criminology is to use simulation to understand the dynamics within criminal networks, in order to find new ways for intervening against such organisations. Police actions in this milieu are a very sensitive topic, since the well-being of humans might be at stake, if such operations are planned without taking every bit of available information into account. Thus, the opportunity to get access to a kind of inside-perspective of a criminal network is most welcome. As this is a new terrain for both criminologists and simulation experts, the simulation approach has to be carefully selected, carried out, evaluated and finally approved. In this context it is particularly important that the simulation considers the domain knowledge and available evidence information by involving relevant stakeholders.

This PhD thesis aims to bring together existing knowledge and methods from computer science and social simulation, adapt these methods for the specific requirements of criminology, and develop simulation software according to these requirements. To scope this work in more details, the subsequent sections set the specific research questions and the research approach.

1.2 Problem scope and research questions

Regularly reappearing debates take place since the early days of computational social science about whether to ground simulation models upon theory or evidence, and about the relation between these two pillars of science in the context of simulation — see sections 2.3.1 and 2.3.3. The question of complexity of the modelled system is often raised in such discussions. In the domain of social simulation, an agreement has been reached that evidence-based simulation is a beneficial approach for particularly complex systems, when supported by the envisaged simulation types and paradigms, and underpinned with the appropriate supporting methods, such as data analysis.

To find the appropriate level of complexity for a concrete model development task, the different perspectives (or roles) of the involved persons have to be taken into account. On the one hand there is the 'user' perspective. Users are typically stakeholders — such as policy owners, decision makers, police investigators — who want to see simple, aggregated results and might want to play with parameters and see the change of results immediately. This raises a number of issues that need to be regarded:

- 'Playing' with a model for non-simulation experts is only possible when the number of parameters is small enough.

- Results of simulation runs have to be validated and analysed, often manually by a simulation expert or analyst.

- For complex and large-scale models, sufficient computing power must be at hand.

This leads to questions on the part of the 'simulation modeller', who is in charge of translating the user requirements into software:

- What are the relevant user requirements in terms of model structure and behaviour, and how can these associated with model parameters?
- What is the right way to implement the requirements in terms of modelling approach and programming style?
- How can the model be verified, i.e. how can it be ensured that it behaves as specified in the requirements?
- How can the model be validated, i.e. how can it be ensured that the simulation reflects the real-world system envisaged by the user?

The simulation modeller has to deal with some tradeoffs these different requirements and questions impose, and needs to find a suitable compromise for the eventual solution. As for every software development project, this involves the search for the appropriate development processes and methods, programming languages and other tools. In the domain of simulation, this process often starts with a fundamental decision between two implementation approaches:

- Should a specialised simulation tool be used? This allows modelling on an abstract (and presumably more understandable for the user) level, but is possibly only feasible to capture limited and specific sets of problems.
- Should a general purpose language be used? This requires higher computer science and coding skills and the code will be hard to understand for users, but complex and multifaceted models are practicable.

In addition, a scope of problems has to be taken into account: the interaction between the two perspectives of the user and the simulation modeller. The modeller has to capture the information about the topics that should be reflected by the simulation model. This is a potential source of errors because of communication problems, often caused by different meanings of the same terms in the different (technical) languages. As the concrete software realising the simulation model typically remains a 'black box' to the user, it is a critical task for the modeller to justify the implementation decisions and even simulation results to the user, and at the same time being aware to identify potential misunderstandings. On the other side, it is also hard for the user to see whether his/her ideas are reflected by the model. Often the expectations towards simulation outcomes are high, while in many cases users are left disappointed when the results do not fully match the expectations. It is an

important and demanding task for the modeller to find out whether a disappointing outcome is due to limited model assumptions, model errors or — most pleasing — the discovery of an unexpected emergent phenomenon.

To solve these issues, the idea of drawing the modelling (and implementation) process on an more abstract level is generally a most worthy cause — like it has been proven successful for typical software engineering tasks, e.g. with model-driven approaches and code transformation. That such approaches are not widely used in social simulation yet has to do with the fact that requirements for models appropriately treating human behaviour usually cannot be defined during an early specification phase in a level of detail necessary to generate code. Often necessary model aspects only become apparent when simulation results are available. Only then it is possible to judge whether the relevant aspects are treated sufficiently, which are missing, and which might even be superfluous. In other words, for many cases of simulation models the developed software generates its own requirements.

Thus, an approach to deal with these problems should try to break open the black box simulation model as far as possible, by making the program behaviour visible without unveiling the program code. Here, the work presented in this PhD thesis — done mostly in collaborative research projects — steps in, with an approach that combines model-driven techniques with traceability.

Objective of this PhD thesis is to tackle some of the involved problem fields from computer science perspective, in combination with the perspectives of criminology and social sciences, as indicated in the introduction of this chapter.

The intention is to make a contribution in order to render the process of creating complex simulation models open and more transparent, and to make the resulting models better understandable for interested stakeholders — and to approach these contributions from different perspectives. Hence, four overarching research questions can be formulated:

> **RQ 1:** How must a modelling process for developing evidence-based social simulation models be designed, taking questions about stakeholder involvement, requirement elicitation, model implementation, verification and validation into account?
>
> **RQ 2:** What are the appropriate methods to be applied within the different modelling process phases that are aligned with the questions referred to in RQ 1?
>
> **RQ 3:** How can the methods according to RQ 2 be supported by software tools?
>
> **RQ 4:** How can process (according to RQ 1), methods (according to RQ 2) and tools (according to RQ 3) be applied for creating a specific simulation model in the domain of criminology?

These four top-level research questions — guiding the research present in this thesis and revisited in the concluding Chapter 11 — provide an overarching view of a technical perspective on the research addressed by the two international cooperation projects introduced in Part I.

With regard to RQ 4, more specific research questions are formulated (and responded to) in Part II.

1.3 Research approach and structure of the PhD thesis

A major part of this PhD thesis is concerned with engineering activities, resulting in specifications and documentations of implemented software. Other than that, the research design — in particular in view of the use case in Part II — contains mainly descriptive and experimental elements. This concerns for example activities to familiarise with application fields, and selecting and applying specific methods of computer science.

Figure 1.1 shows the parts and components that constitute the overall research approach, which are the basis for the work presented in this thesis. On the top of the diagram, the four main pillars are exposed: process design, theoretical grounds, software tools and application.

The *process design* relates to a simulation model development process, comprising six phases drawn (from top to bottom) in the first column. In particular the simulation modelling and experimentation phases are focussed in this work. This process is presented in Part I of this PhD thesis.

Each of the process phases is supported by tools based on some *theoretical grounds*. The concrete approaches used in this work are identified in the second column, with most relevant subjects of agent-based simulation, declarative rule engine and visualisation (encapsulating approaches for processing and presentation) for both results from simulation experiments and for evidence traces. In Chapter 2, definitions and information on the state-of-the-art are provided.

For each of the phases, specific *software tools* are mentioned that are based upon the theoretical grounds. These tools are outlined in Part I in combination with the process description. However, three tools listed here — DRAMS, together with two plugins for UI and Model Explorer — constitute one of the main contributions of this work. Detailed information and specifications for these tools are contained in Part III.

Finally, a simulation model that serves as an *application* example is sketched in the fourth column. The entire process of conceptualising and implementing this model is the second main contribution of this work, presented in Part II.

Summarising, this PhD thesis is structured in three parts which are also reflected in Figure 1.1.

Part I outlines the frame in which the contributions of this PhD thesis are

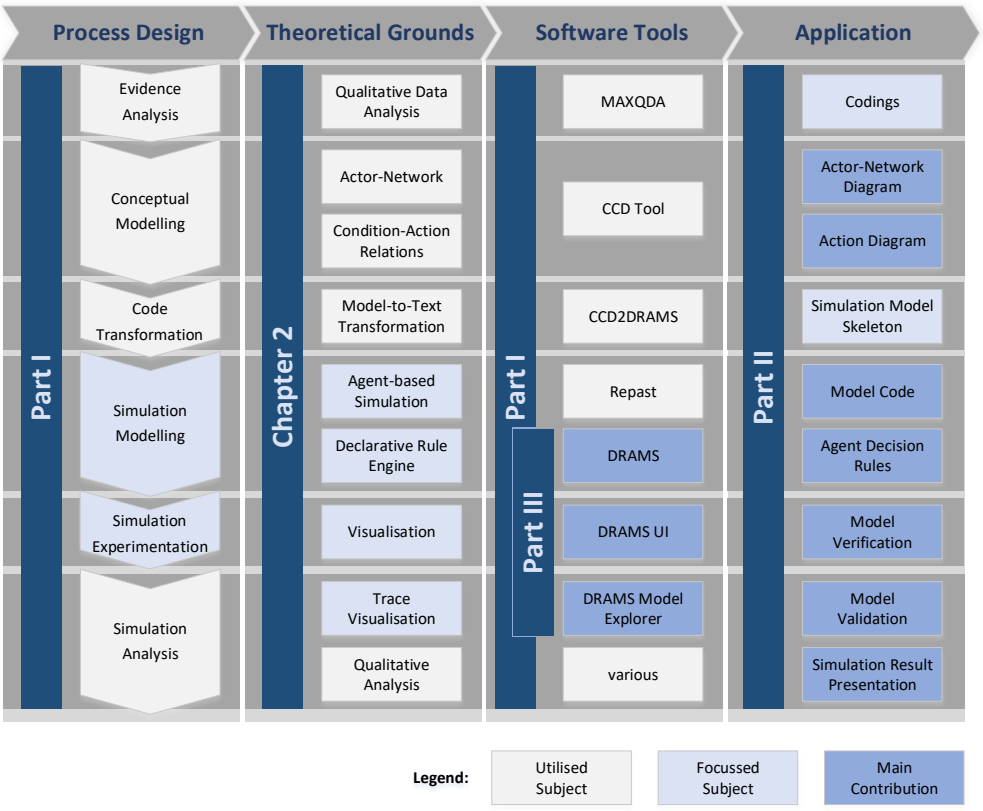

Figure 1.1: Parts and components of the research approach

fitted. This part contains two chapters: Chapter 3 provides an overview on the model development process and tools for evidence-based policy modelling, while Chapter 4 shows a customisation of this process.

Part II demonstrates how the processes and tools described in Part I are applied, concentrating mainly on computer science relevant topics. In three chapters the use case is introduced and conceptualised (Chapter 5), then implemented (Chapter 6), and finally simulation results are presented and interpreted (Chapter 7).

Part III then complements the contents of part Part I and II with a deeply technical view on the used simulation tool. This part contains three chapters: Chapter 8 gives an introduction to DRAMS, Chapter 9 provides the related technical specifications, while Chapter 10 discloses the technical features of traceability.

Chapter 2

Foundations: Terminology and State-of-the-art

2.1 Introduction

This chapter defines the key concepts targeted by this PhD thesis. The title 'Engineering Criminal Agents' already indicates the three main research areas this work is based upon: Software Engineering, (Agent-based) Simulation and the application field of Social Sciences, of which the specific focus is on aspects of crime. The following three sections clarify the terminology for each of these research fields as understood and used throughout this PhD thesis, where feasible drawing on standardised definitions[1]. In conjunction overviews on the state-of-the-art are given. The organisation of these parts is top-town: from the overarching concept to the specific topics relevant for this work.

2.2 Software Engineering

Software engineering as a discipline encompasses a variety of aspects. The following definition of this term gives a coarse overview of the field:

> "The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software." (IEEE, 1990, p. 70)

Originating from this definition, a set of sub-disciplines can be distinguished which concretise the different aspects. From the definition, three disciplines can directly be derived:

[1] Like given e.g. by the IEEE — also in order to appreciate some 'stable pillars' over time in this otherwise quickly changing domain.

> *Software development*: "The process by which user needs are translated into a software product. The process involves translating user needs into software requirements, transforming the software requirements into design, implementing the design in code, testing the code, and sometimes, installing and checking out the software for operational use. Note: These activities may overlap or be performed iteratively." (IEEE, 1990, p. 70)

> *Software operation*: "The process of running a computer system in its intended environment to perform its intended functions." (IEEE, 1990, p. 53)

> *Software maintenance*: "The process of modifying a software system or component after delivery to correct faults, improve performance or other attributes, or adapt to a changed environment." (IEEE, 1990, p. 47)

This work primarily focusses on the software development discipline, while software operation and maintenance play a subordinate role. Throughout this PhD thesis, the development of two very dissimilar software systems will be exhibited in detail. The differences do not only regard the application and the applied programming paradigm, but also the actual development process.

The foundations elaborated in the following subsections apply to either both or to one of the two software systems. Starting with models of development processes, the focus is then turned towards methods for collecting requirements (and in general data to inform the software system), to transforming the requirements into conceptual models and ultimately into program code while keeping track of the transformation steps via traceability, and concludes with deliberations on software testing. By providing a common view on these aspects, it becomes possible to classify the two systems within this framework.

2.2.1 Software processes and process models

Process models in the context of this PhD thesis define types of systematic approaches to support the software development process as defined above. As defined by Sommerville,

> "Software processes are the activities involved in producing a software system. Software process models are abstract representations of these processes." (Sommerville, 2010, p. 53)

> "Process models may be developed from various perspectives and can show the activities involved in a process, the artifacts used in the process, constraints that apply to the process, and the roles of the people enacting the process." (Sommerville, 2010, p. 742)

There exist countless different process models, ranging from generic standardised process models, to models designed for specific purposes. Examples for the former type are:

- The *waterfall model* "in which the constituent activities, typically a concept phase, requirements phase, design phase, implementation phase, test phase, and installation and checkout phase, are performed in that order, possibly with overlap but with little or no iteration." (IEEE, 1990, p. 86).
- In the *incremental development* the activities defined in the waterfall model are interleaved and the "system is developed as a series of versions (increments), with each version adding functionality to the previous version." (Sommerville, 2010, p. 30).
- *Reuse-oriented software engineering* "is based on the existence of a significant number of reusable components. The system development process focuses on integrating these components into a system rather than developing them from scratch." (Sommerville, 2010, p. 30).

Detailed description of the models are provided e.g. by (Sommerville, 2010, p. 29–36).

A more specific process model (and as such an example of the latter type mentioned above) is the OCOPOMO process. This will be introduced below, further explained in Chapter 3 and applied as a concrete process in Chapter 4 and Part II of this PhD thesis.

Two kinds of scenarios can be distinguished in which process models can be instantiated:

- On the one hand, *plan-driven development* "identifies separate stages in the software process with outputs associated with each stage. The outputs from one stage are used as a basis for planning the following process activity." (Sommerville, 2010, p. 62)
- On the other hand, *agile development processes* "consider design and implementation to be the central activities in the software process. They incorporate other activities, such as requirements elicitation and testing, into design and implementation." (Sommerville, 2010, p. 62)

Over the past two decades, incremental and reuse-oriented software processes have gained popularity. In particular in combination with rapid or agile software development these became increasingly important. More specifically (and as a distinction to 'purely' rapid development), agile development processes are "incremental development methods in which the increments are small [...]. They involve customers in the development process to get rapid feedback on changing requirements. They minimize documentation by using informal communications rather than formal meetings with written documents." (Sommerville, 2010, p. 58). Important contemporary process models

for rapid and agile development are for example included[2] in rapid prototyping and the Scrum framework, respectively.

According to (Sommerville, 2010, p. 45), prototyping is the "rapid, iterative development" of a prototype, "an initial version of a software system that is used to demonstrate concepts, try out design options, and find out more about the problem and its possible solutions". Prototypes can "help with the elicitation and validation of system requirements" and for example "used to explore particular software solutions and to support user interface design". "They may get new ideas for requirements, and find areas of strength and weakness in the software". Prototyping can also be employed for the development of mature software, e.g. with the approach of evolutionary prototyping (Sherrell, 2013).

A representative of the agile approach is the Scrum framework. The term 'scrum' was first used by Takeuchi and Nonaka (1986) and was borrowed from sports: "Like a rugby team, the core project members [...] stay intact from beginning to end and are responsible for combining all of the phases" (Takeuchi and Nonaka, 1986, p. 6). This allegory is just one assertion of a framework property, while Schwaber (2004) provides a more practical definition: Scrum is "devised specifically to wrest usable products from complex problems", "based in industrial process control theory, which employs mechanisms such as self-organization and emergence". The core of Scrum is an "iterative, incremental process skeleton". Iterations — also called sprints — consist "of development activities that occur one after another. The output of each iteration is an increment of product." In each iteration, regular ("daily") inspections are included "in which the individual team members meet to inspect each others' activities and make appropriate adaptations. Driving the iteration is a list of requirements. This cycle repeats until the project is no longer funded." This process skeleton is surrounded by "an outline planning phase" as input stage, where "the general objectives for the project and design the software architecture"(Sommerville, 2010, p. 72) are established, and a final "project closure phase" that "wraps up the project, completes required documentation such as system help frames and user manuals, and assesses the lessons learned from the project" (Sommerville, 2010, p. 73). In contrast to other agile approaches such as Extreme Programming (XP; Beck (1999), Beck and Andres (2004)), "Scrum does not prescribe the use of programming practices such as pair programming and test-first development" (Sommerville, 2010, p. 72).

The OCOPOMO process model as a derivative and specialisation of the incremental development process model serves as an example for the more specific type as mentioned in the beginning of this section. 'More specific' implies that activities and other aspects within the process are defined with a particular purpose and, hence, are laid out in more detail. These aspects are characterised in (Bicking et al., 2010, p. 58–76):

- In the *scenario building process* the policy areas are identified and analysed, using the scenario method. The scenarios developed in this phase

[2]The process model aspect is just a specific perspective on such frameworks.

are part of the requirement set for subsequent development of policy models. Further details are given in the following subsection Requirement Analysis.

- *Integrating stakeholder-generated scenarios and formal policy models* are the stages of formalisation, firstly into conceptual models, then into formal simulation models. Several aspects linked to this topic are elaborated in the subsections Conceptual Modelling, Implementation support and Traceability.

- In the *policy modelling process* a formalisation of the conceptual models is realised. This is subject of subsection Social simulation.

- *Participation processes* are associated with the other phases listed above. Subsection Participation adds some thoughts to this topic.

From this process model the OCOPOMO process was derived and disseminated e.g. in (Scherer et al., 2013b) and (Scherer et al., 2015). The prominent feature of the OCOPOMO process is to bring together the need for an agile approach and prototyping for model development, with a surrounded plan-driven approach required for participation and structured requirement analysis. Chapter 3 is dedicated to this topic; an outline of the process and further references can be found there.

2.2.2 Requirement Analysis

"The requirements for a system are the descriptions of what the system should do — the services that it provides and the constraints on its operation." (Sommerville, 2010, p. 83) Different levels of requirements are distinguished, where *user requirements* refer to high-level assertions formulated in natural language about the expected software behaviour, and *system requirements* cover technical details about the "software system's functions, services, and operational constraints" (Sommerville, 2010, p. 85). A further distinction is made into *functional* and *non-functional* requirements, where the former refers to the actual functional of the software, while the latter covers constraints of the software like usability and performance. Apart from Sommerville (2010), e.g. Wiegers and Beatty (2013) provides a comprehensive overview on this topic and associated aspects like proper notation and documentation of requirements.

Requirements are gathered, evaluated and documented in the requirement analysis process. This is defined in (IEEE, 1990, p. 65) as:

> "(1) The process of studying user needs to arrive at a definition of system, hardware, or software requirements. (2) The process of studying and refining system, hardware, or software requirements."

Requirement analysis is part of the software specification or requirements engineering, which "is the process of understanding and defining what services are required from the system and identifying the constraints on the system's operation and development" (Sommerville, 2010, p. 36). It might be preceded by feasibility study, and succeeded by requirement specification and validation (Sommerville, 2010, p. 37–38).

Of particular interest for this work are specific methods for requirement gathering and analysis: one the one hand the scenario studies, on the other hand data analysis.

A *scenario study* is an often applied method for describing situations and perspectives for strategic planning[3] (Codagnone and Wimmer, 2007). As defined by Carroll (1995), scenarios are narratives which describe and, thus, allow to explore possible future states of some system. They allow to capture perspectives of different stakeholders in a language familiar to them. A scenario can for example describe "a related sequence of events which identifies exogenous factors determining which events occur and also how some of these events cause or modify the nature of subsequent events" (OCOPOMO consortium, 2012). The scenario method is in particular useful to elicit requirements for policy models, where effects of different political decisions should be investigated (Bicking et al., 2010).

Requirements can also be elicited from available documents or any kind of data repositories, both of structured or unstructured nature. In cases where the amount of data is not reasonably manageable by 'desk research' studies, data analysis methods come into play. A distinction is made between quantitative and qualitative data analysis, which goes back to the (fundamental or blurred, depending on definition; Given (2008)) distinction between quantitative and qualitative research methods — although the combinations of quantitative and qualitative research have been investigated, e.g. by Bryman (2006).

Quantitative methods in general are based on "approaches to empirical inquiry that collect, analyze, and display data in numerical rather than narrative form" (Given, 2008, p. 713). "The original data can be in nonnumerical form such as statements that are recoded on some specific numerical scale" (Given, 2008, p. 185). In the context of this work, quantitative data analysis is linked to text mining (see Aggarwal and Zhai (2012) and Diesner and Carley (2005)), which "is often studied as a separate subtopic within data mining" (Aggarwal, 2015, p. 9) with specialised methods for data preparation (including tokenisation and stemming, see Aggarwal and Zhai (2012)), clustering and classification. According to Aggarwal (2015), the data mining process is a pipeline containing several phases such as data collection, feature extraction and data cleaning (subsumed under the data preparation phase), and an analytical phase where algorithms specific to the data mining application are applied. An application example of a quantitative analysis based on text min-

[3] `http://www.egovrtd2020.org/navigation/work_packages/wp2_scenario_building/`

ing on data relevant for Part II of this PhD thesis is provided by Neumann and Sartor (2016) and Sartor (2015); a broader context is given in (Sartor et al., 2014).

Qualitative methods, on the other hand, are "not primarily quantitative (numerical) in nature" and are able to "embrace the quality or essence of something, some phenomenon, or even some event". "Qualitative research places emphasis on understanding through looking closely at people's words, actions and interactions, and traces or records created by people. Qualitative research examines the patterns of meaning that emerge from systematic observations of people's words, actions and interactions, and traces or records" (Given, 2008, p. 826). Qualitative data analysis is understood here as a computer-assisted text-analysis process. Hence, software distributed under the label CAQDAS[4] is the means of choice. In section 4.4, a concrete example process for applying qualitative data analysis for requirement elicitation for a simulation model software (in conjunction with conceptual modelling, see next section) is given.

Other examples for exploiting data analysis for requirement analysis are for example given by Edmonds (2015b) and Edmonds (2015a) in the domain of agent-based social simulation, by Cleland-Huang and Mobasher (2008) for requirement elicitation in large-scale software projects, by Hayes et al. (2005) on how text mining supports analysts in software engineering tasks, and by Galvis Carreño and Winbladh (2013) who propose topic modelling for extracting requirements from user feedback. Taylor and Giraud-Carrier (2010) give an overview on applications of data mining in software engineering.

2.2.3 Conceptual Modelling

As stated by Thalheim (2011) conceptual modelling "aims to create an abstract representation of the situation under investigation" (Thalheim, 2011, sect. 1.5), in this context the specification of a software system. "Conceptual models enhance models with concepts that are commonly shared within a community or at least between the stakeholders involved in the modelling process. [...] Conceptualisation aims at collection of objects, concepts and other entities that are assumed to exist in some area of interest and the relationships that hold among them." (Thalheim, 2011, sect. 1.5) In particular for simulation models as the special case of software systems relevant for this PhD thesis (and discussed in section 2.3), (Robinson, 2008, p. 283) proposes that a conceptual model is "a non-software specific description of the computer simulation model [...], describing the objectives, inputs, outputs, content, assumptions and simplifications of the model". Different perspectives on the definition of conceptual modelling for simulation are collected by (Robinson et al., 2015, p. 2823).

Proposed purposes of conceptual models in the context of software engineering range between two 'extremes':

[4]Computer-Assisted Qualitative Data Analysis Software

- On the one end, to maintain a (documented) conceptual model for communication purposes (Robinson et al., 2015, p. 2823).

- On the other end to program with conceptual models, i.e. "programming activities are to be carried out via conceptual modelling" where for "applications amenable to conceptual-model designs, software developers should never need to write a line of traditional code" (Embley et al., 2011, p. 3).

A perspective in-between these extremes is adopted for the context of this PhD thesis: The purpose (and expected benefit) of a conceptual model is to foster the understanding of the "system under investigation" and support its implementation into software. Such conceptual models firstly "must be sufficiently transparent so that all stakeholders [...] are comfortable in using it as a means for discussing those mechanisms within the [system under investigation] that have relevance to the characterization of its behavior" (Robinson et al., 2015, p. 2814). Secondly, the "conceptual model must be sufficiently comprehensive so that it can serve as a specification for developing a computer program" (Robinson et al., 2015, p. 2815).

The purpose of a conceptual model to represent a software specification or even an executable formalisation requires some form of (automated) transformation from the abstraction level of concepts to executable code. Depending on the degree of formalisation on the side of the conceptual model, either parts of program code, or an entire executable program can be generated. Such transformation and code generation approaches are subsumed under labels such as Model Driven Architecture (MDA) and Model Driven Development (MDD). Liddle (2011) brings these techniques in relation to conceptual modelling; a technical view is provided in subsection Implementation support.

Conceptual models can be formulated in different languages. Popular approaches are:

- The Universal Modelling Language (UML, Booch et al. (2005)) — in particular the class diagram — is the "most advanced and most widely used language for information modeling" (Robinson et al., 2015, p. 2823). In relation with the Object Constraint Language (OCL; also part of UML) the formal constraints can be specified in a way sufficient to generate executable programs (Liddle, 2011).

- An Ontology "defines a set of representational primitives with which to model a domain of knowledge or discourse" in a language "closer in expressive power to first-order logic". The entities ("representational primitives" — mainly classes, attributes and relationships) of the ontology "include information about their meaning and constraints on their logically consistent application". Ontologies are used for knowledge representation, e.g. to "specify standard conceptual vocabularies in which to exchange data among systems, provide services for answering queries,

> publish reusable knowledge bases, and offer services to facilitate interoperability across multiple, heterogeneous systems and databases". (Gruber, 2009)

Another simulation-specific approach with references to both above-mentioned approaches is the Consistent Conceptual Description (CCD), developed in the OCOPOCMO project (Scherer et al., 2015). The CCD contains both a static perspective (in form of an actor-network-diagram, a graphical language to describe actor-networks; Latour (2005)) and a dynamic perspective (in form of a condition-action diagram, a kind of flow chart with interlinked actions and related pre- and postconditions). Details are given in sections 3.3.2 and 5.4 of this PhD thesis. The CCD can serve both as a communication medium between stakeholders of different domains in participation activities e.g. for policy modelling, and as basis for code generation of simulation models[5]. As applied in the domain of social simulation, the CCD provides a link between *humans* as 'active entities' in the real world and *agents*[6] as software programs representing specific aspects of humans (according to Gilbert and Troitzsch (2005)), by introducing the *actor* concept, an abstraction of the human, equipped with restricted cognitive and behavioural abilities and embedded in a restricted and structured world.

In the domain of agent-based simulation, conceptual modelling (and software engineering in general) has not been widely used until tools like Repast Simphony (North et al., 2013) came into market (Scherer et al., 2015). This has evolved in the recent years, like shown in the various other papers (apart from (Scherer et al., 2015)) summarised by Siebers and Davidsson (2015), or included in an earlier collection on agent-oriented software engineering by Müller and Cossentino (2013): a "methodological approach to model driven design of multiagent systems" by Fischer and Warwas (2013), an "MDA-based approach for implementing secure mobile agent systems" by Kallel et al. (2013) or "'Engineering' Agent-Based Simulation Models?" by Klügl (2013). In addition, contributions by Cetinkaya (2013), Cetinkaya and Verbraeck (2011), Cetinkaya et al. (2011) show progress with respect to "defining and transforming conceptual models into simulation models by using metamodels and model transformations".

Apart from the simulation domain, further application fields of conceptual modelling of structures, processes, user interfaces and other special applications are thoroughly discussed in (Embley and Thalheim, 2011). More recent examples of research in this field are contained in collections on conceptual modelling by Mayr et al. (2017) and Trujillo et al. (2018), where e.g. Delcambre et al. (2018) cast existing research into a reference framework, or where e.g. Evertsz et al. (2017) address applications such as conceptual modelling of

[5] A full specification of program code constraints is not possible with the CCD version as of 2018.

[6] Actors in conceptual models of social systems cannot only become agents in formal models (in ABS), but also e.g. equations (in cellular automata or system dynamics).

multi-agent decision-making within teams.

2.2.4 Participation

In the Oxford English Dictionary participation is defined as the "action of taking part in something". This generic definition can be established in many domains and ways. Participation in the context of this PhD thesis is seen as a backing feature in different processes and activities in relation with requirements engineering and conceptual modelling (as discussed in the two previous subsections). Thereby, stakeholders and domain experts participate in specification, conceptualisation and implementation of software systems.

On the one hand, e-Participation in a sense as described by (Scherer, 2016, pp. 13–16) can contribute to the development of policy models like in the OCOPOMO project (see Chapter 3), by applying information and communication technologies (ICT) in definition of policy cases, collaborative foresight scenarios and elicitation of requirements for respective (conceptual) policy models (Bicking et al., 2010) and technical specification and implementation architecture (Scherer, 2016, p. 113).

On the other hand, the approach of participatory modelling is relevant in the process of designing and evaluating a model (Voinov and Bousquet, 2010) of a use case in criminology as outlined in Chapter 4, where teamwork of a modelling expert group (with roles Data Analyst and a Simulation Modeller as described in section 5.5) is accompanied with participation of police stakeholders as field experts — mainly by means of informal discussions. Participatory modelling can be more formally defined as an "approach combining participatory procedures with modeling techniques" with the purpose to establish and effectively work towards a "collective vision for managing a common resource". It "integrates different points of view and representations of reality through collectively building a common model." (Jones et al., 2009).

Related works include e.g. a "companion modelling approach" outlined in (Barreteau, 2003), and the integration of MDA with agent-based modelling (ABM) by applying executable UML activity diagrams presented in (Bommel et al., 2014). Apart from ABM, Alvertis et al. (2016) propose an approach to "align software development with market expectations throughout the software development cycle", using a "social collaborative development platform" that involves "end users in every phase of the software development process".

2.2.5 Traceability

Traceability as a feature of software engineering has various different meanings, with some accumulation over time. Two long-established definitions of the term *traceability* are provided in (IEEE, 1990, p. 82):

> "(1) The degree to which a relationship can be established between two or more products of the development process, especially

> products having a predecessor-successor or master-subordinate relationship to one another; for example, the degree to which the requirements and design of a given software component match."
>
> "(2) The degree to which each element in a software development product establishes its reason for existing; for example, the degree to which each element in a bubble chart references the requirement that it satisfies."

Traceability is manifested in the creation of *traces*, for which (IEEE, 1990, p. 82) provides three definitions:

> "(1) A record of the execution of a computer program, showing the sequence of instructions executed, the names and values of variables, or both. Types include execution trace, retrospective trace, subroutine trace, symbolic trace, variable trace."
>
> "(2) To produce a record as in (1)."
>
> "(3) To establish a relationship between two or more products of the development process; for example, to establish the relationship between a given requirement and the design element that implements that requirement."

Definition (2) of traceability, together with definition (3) of trace, best capture the meaning relevant for this PhD thesis. Traceability in this context is the generation and preservation of traces from different artefacts generated in the model and software development and execution process, in order to ensure provenance of these artefacts (Lotzmann and Wimmer, 2013a). In fact, evidences provided in requirements analysis and participatory modelling can be tracked from the artefacts generated in the formalisation (both conceptual modelling and implementation) and from outputs of the implemented software, in this case an agent-based simulation model. This particulate approach is detailed in section 3.4, exemplified in Part II and underpinned from a technical perspective in Chapter 10 of this PhD thesis.

A similar approach (i.e. using qualitative research to inform simulation models) is studied by Polhill et al. (2010), where the authors pointing out the significance of making this kind of provenance explicit, drawing on work by Taylor (2003).

The importance of traceability has been on the research agenda for quite some time. Gotel and Finkelstein (1994) performed an empirical study on the traceability of requirement lifetime in industrial engineering projects, pointing out that not only detecting, obtaining and recording relevant information, but also the organisation, maintenance, access and presentation of this information are crucial for requirement traceability and need further research. Sommerville (2010) digs into these issues as part of software requirement engineering and provide a rationale for requirements traceability:

> "You need to keep track of the relationships between requirements, their sources, and the system design so that you can analyze the reasons for proposed changes and the impact that these changes are likely to have on other parts of the system. You need to be able to trace how a change ripples its way through the system." (Sommerville, 2010, p. 114)

A practical evaluation on the topic is done by Wiegers and Beatty (2013), who dedicate a chapter to "links in the requirement chain".

Another perspective on traceability and simulation is studied by Venturini et al. (2015): In case of structured data like in contemporary electronic or social media traces are generated in the course of data recording. Such traces can itself inform simulation models to capture e.g. "more realistic ways how people change opinion".

2.2.6 Implementation support

This and the next subsection cover implementation-specific aspects of software engineering, partly taking up again issues introduced in the previous subsections and, thus, adding a technical perspective for supporting the software development process.

This subsection firstly elaborates on possible ways to come from conceptual models to executable code. Point of departure are Model-driven Architecture (MDA) and Model-driven Development (MDD) approaches introduced in subsection Conceptual Modelling. The official MDA guide by the Object Management Group (OMG) defines:

> "MDA provides an approach for deriving value from models and architecture in support of the full life cycle of physical, organizational and I.T. systems." (Object Management Group (OMG), 2014, p. 1)

In the MDA approach, models are seen as "communication vehicles" which at the same time allow a "fully or partially automated" derivation of artefacts and implementations. This "derivation involves a platform independent '*source model*' with some parameters for how that source model is to be interpreted, a '*transformation*' and a platform specific '*target artifact*"' (Object Management Group (OMG), 2014, p. 2).

Source models are the conceptual models discussed before. As target artefacts program code is considered here, primarily in form of code stubs. According to Scherer et al. (2013b), necessary prerequisite for a transformation is the existence of a meta-model at least for the source model. For the target artefact either a meta-model needs to be present, or an appropriate language specification (potentially together with code templates). In the former case a model-to-model transformation can be performed (as exemplified e.g. by

Magalhães et al. (2016)), in the latter case 'only' a model-to-text transformation is possible, in a way as outlined in section 3.3.3 and detailed in (Lilge, 2012), with CCD as source model and an executable simulation model as target artefact. A benefit of model-to-model transformation (over the model-to-code approach) is that a bi-directional transformation is possible. This means that if the target artefact is changed, then the changes are reflected in the source model, and vice versa. The latter is the only feasible case for model-to-text transformation.

Recent research focusses on broadening the application range of MDA. E.g. (Pastor, 2017) show how MDA can be integrated in Capability-driven Development (CDD), an approach that covers business process modelling, requirements engineering and object-oriented conceptual modelling. A quite topical literature review on MDA has been issued by Elsawi et al. (2015).

Especially in the early days of MDA, the approach also received critical responses: an essay by (Reeves, 2005) and a paper by (Picek and Strahonja, 2007) who raises the question about "Future or Failure of Software Development" should be mentioned here.

Another aspect relevant to possible program code target artefacts in MDA — but also independently from this topic — is the programming paradigm in which this program code is formulated. In the context of this PhD thesis the differentiation between imperative and declarative programming languages is of relevance.

> *Imperative programming language:* "A programming language in which the user states a specific set of instructions that the computer must perform in a given sequence." (IEEE, 1990, p. 59)

> *Declarative programming language:* "A nonprocedural language that permits the user to declare a set of facts and to express queries or problems that use these facts." (IEEE, 1990, p. 22)

While many contemporary and commonly used programming languages — such as Java[7], C++[8], C#[9] — follow the imperative paradigm (and increasingly make use of other concepts, like adding declarative language elements such as the functional 'lambda expression' to Java; Subramaniam (2014)), declarative languages have some important niche applications, mainly in the (broad) context of artificial intelligence. One such application is the declarative rule engine (such as Jess; Friedman-Hill (2003), used for example in expert systems Leondes (2002) or for providing rule sets to autonomous software ('agents', as described e.g. by Scherer et al. (2015), or realised by the SDML[10] agent programming language, see Moss et al. (1998); refer also section 2.3).

[7] `http://oracle.com/java/`

[8] `https://isocpp.org/`

[9] `https://docs.microsoft.com/dotnet/csharp/language-reference/`

[10] Strictly Declarative Modelling Language

2.2.7 Software testing

Software testing is a crucial ingredient in any kind of software development process. The following definition is contained in (IEEE, 1990, p. 80):

> *Testing:* "The process of analyzing a software item to detect the differences between existing and required conditions (that is, bugs) and to evaluate the features of the software items."

A comprehensive overview on the topic is given by Sommerville (2010), including the different phases of the software lifecycle where testing has to be applied (i.e. development, release or user testing), with the suitable methods to use (i.e. unit, component or system testing as a top-level categorisation). "Testing is part of a broader process of software verification and validation" (Sommerville, 2010, p. 206). These two terms can be defined as follows:

> *Verification:* "The process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase." (IEEE, 1990, p. 85), or more informal:
> "Am I building the product right?" (Boehm, 1979, p. 711)

> *Validation:* "The process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements." (IEEE, 1990, p. 85), or more informal:
> "Am I building the right product?" (Boehm, 1979, p. 711)

These topics will be revisited at different places in this PhD thesis, e.g. in section 8.4.2 and — regarding the concrete realisation of verification and validation of simulation models — in sections 7.3 and 7.5.

2.3 Simulation

The term simulation is part of many professional and scientific terminologies, but also of everyday language. Simulation can be used for many different purposes. Gilbert and Troitzsch (2005) list six typical fields of usage:

- for *prediction* — e.g. providing forecasts for economic developments,
- for *training* — e.g. in form of flight simulators,
- for *entertainment* — e.g. in flight simulation computer games,
- to *substitute* human capabilities by e.g. expert systems,
- to *discover* consequences of theories e.g. in physics or social sciences, and

- to *understand* features of a system, as a research method e.g. in physics or social sciences.

While actual meaning and semantic scope widely differ from one application field to another, most of these share the aspects expressed in the following definitions:

> "Simulation is the imitation of the operation of a real-world process or system over time." (Banks, 1998, p. 3)

> "A simulation is an applied methodology that can describe the behavior of that system using either a mathematical model or a symbolic model." (Sokolowski and Banks, 2009, p. 5), citing Fishwick (1995)

> "A model that behaves or operates like a given system when provided a set of controlled inputs." (IEEE, 1990, p. 69)

A definition of simulation as a research method — as adopted in this PhD thesis — is provided by Gilbert and Troitzsch (2005):

> "Simulation is a particular type of modelling. Building a model is a well-recognized way of understanding the world: something we do all the time, but which science and social science has refined and formalized. A model is a simplification — smaller, less detailed, less complex, or all of these together — of some other structure or system." (Gilbert and Troitzsch, 2005, p. 2)

Although the definitions of the term simulation given in the beginning of section 2.3 leave room for any kind of model of any kind of system, it is nowadays typically associated with information systems: simulation models are regarded as software systems. "Advancements in computer software and hardware [...] have hastened the pace of the maturation of [modelling and simulation] as a discipline and tool." (Sokolowski and Banks, 2009, p. 13)

Computer simulation models exist in many different appearances. Sokolowski and Banks (2010) provide a classification scheme that distinguishes between simulation paradigms, attributes, verification and validation process, and model types.

Prominent *simulation paradigms* are the Monte Carlo method, continuous simulation and discrete-event simulation. The *Monte Carlo method* models system behaviour using probabilities. The output of the model is repeatedly calculated from random samples from input variables within their value range and distribution, "until there is a sense of how the output varies given the random input values". In *continuous simulation* "system variables are continuous functions of time", expressed by differential equations. In contrast, system variables are discrete functions of time in *discrete-time simulation*. A

special case of the latter is the *discrete-event simulation*, where the system variables change on the occurrence of events, with (implicit) time advancing from one event to another. The change of system variables over time defines such simulation models as dynamic systems. (Sokolowski and Banks, 2010, p. 12)

Attributes define characteristics of the model or simulation. Sokolowski and Banks (2010) distinguish between fidelity, resolution and scale. All three attributes are ordinally scaled e.g. with values 'low' and 'high'. *Fidelity* describes how closely the model matches the real-world system. *Resolution* (or granularity) "is the degree of detail with which the real world is simulated". *Scale* refers to the size of the real-world scenario represented by the simulation. (Sokolowski and Banks, 2010, p. 13)

For the process of *verification and validation* Sokolowski and Banks (2010) provide a table with established techniques, giving an impression on the diversity of simulation approaches, requirements and application domains for which models need to be verified and validated. This selection is reproduced in Table A.1 in the annex. A discussion on these topics is given in subsection Simulation and Software Engineering.

Finally, of the numerous existing *model types*, the following list represents a selection of the most commonly used ones:

- *Physics-based models* are mathematical models where the "equations are derived from basic physical principles". These equations are in many cases models itself, since "many physics-based models are not truly things" but representations of physical phenomena. (Sokolowski and Banks, 2010, p. 16)

- *Finite element models* are decompositions of complicated systems into small elements, which are then modelled often using physics-based equations. The relations between these elements (or nodes) are represented by a mesh of nodes, where the state of each node is dependent on the state of the neighbouring nodes. Such models are widely used for simulation in engineering. (Sokolowski and Banks, 2010, p. 17)

- *Data-based models* rely on "data describing represented aspects of the subject of the model". Data sources "can include actual field experience via the real-world or real system, operational testing and evaluation of a real system, other simulations of the system, and qualitative and quantitative research, as well as best guesses from subject matter experts". Such models are useful when "the real system cannot be engaged or when the subject of the model is notional" or not well understood. (Sokolowski and Banks, 2010, p. 18)

- *Agent-based models* follow the idea "of a computer being able to create a complex system on its own by following a set of rules or directions and not having the complex system defined beforehand by a human". The

modelled entities are agents, "autonomous software entities that interact with their environment or other agents to achieve some goal or accomplish some task". Agent-based models are commonly used for investigating human and social phenomena. (Sokolowski and Banks, 2010, p. 18) "Agent-based modelling studies the interaction of individual agents on a microscopic level (Squazzoni et al., 2014), in relation between cognition and interaction (Nardin et al., 2017)" (Lotzmann and Neumann, 2017).

- *Aggregate models* focus on a potentially large number of rather small objects and actions, represented in a combined manner. Such models are means of choice when the performance of the aggregate matters, mostly in applications where people interact with the simulated system (such as battlefield simulation). (Sokolowski and Banks, 2010, p. 19)

- *Hybrid models* are built using combinations of more than one modelling paradigm. Although more difficult to create models, this approach is widely applied. (Sokolowski and Banks, 2010, p. 19)

Many factors play a role for selecting an appropriate simulation approach. The specific application domain, the existing foundation (in terms of theories or data), the desired kind of simulation results, the technical capabilities and, most importantly, the purpose of the simulation (i.e. the envisaged questions the simulation should answer) need to be taken into consideration.

As indicated at the beginning of this section, purpose and role of simulation can be manifold. In research, simulation is considered an "extremely useful avenue of scientific pursuit, along with theory and experimentation" (Sokolowski and Banks, 2009, p. 25). This research perspective is illustrated in Figure 2.1, derived from an illustration originally introduced by Gilbert and Troitzsch (2005) with an explicitly added simulation software artefact (inspired by (Troitzsch, 2004), citing Drogoul et al. (2003)). At the centre is the real-world system of interest from which a (conceptual) model is created in an abstraction process. This model is implemented in a simulation software that is executed to generate data. This simulated data can be compared for similarity with data gathered from the real-world system.

As a side note, the colours used in Figure 2.1 represent the basic colour scheme used in rich pictures and most diagrams[11] throughout this PhD thesis: no colour (or gray) denote entities of the real world, orange represents data, blue conceptual and green implementation-specific aspects of simulation[12].

This particular notion of simulation is the predominant focus of this PhD thesis, applied for discovery and understanding of social-scientific problems, relying on the agent-based model type. Applying the further classification dimensions from above, the type of model considered here is a dynamic system of the discrete-time paradigm, with potentially high fidelity, and typically a

[11] unless noted differently

[12] In other diagrams, yellow is used in addition to denote theories.

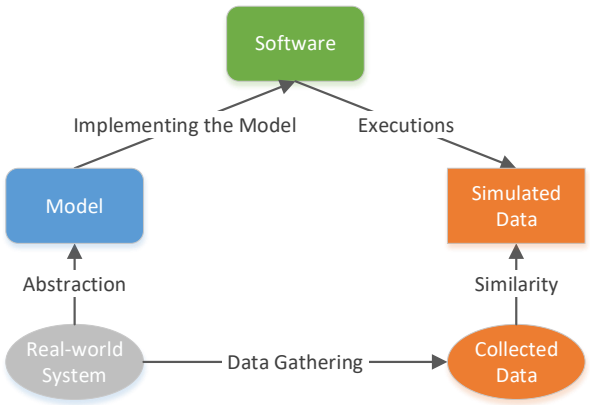

Figure 2.1: Logic of simulation as a research method, according to Gilbert and Troitzsch (2005) with modification inspired by Drogoul et al. (2003)

mutual dependency between resolution and scale (due to computing requirements): high-scaled models have rather low resolution, while low-scaled models might feature a high resolution. (Sokolowski and Banks, 2010, p. 14)

In particular the software aspect is examined in this PhD thesis from two perspectives:

1. Development of a simulation tool.

2. Development of a simulation model using the simulation tool from 1.

The following subsections are dedicated to specific simulation-related topics relevant to this PhD thesis: starting with introductions to the application domain of social simulation, agent-based modelling and evidence-based simulation, and rounded off with an overview on software-related aspects of simulation.

2.3.1 Social simulation

As defined by Silverman et al. (2014), "*social simulation* uses computational methods to examine specific elements of social systems". These computational methods allow to circumvent the problem for social scientists to be "limited by the inextricability of the subject of their research from its environment [...] by generating scenarios inside a virtual environment". Simulations allow "to dynamically and interactively explore [...] ideas and existing knowledge", without compromising the safety of the subjects of research or facing ethical issues. (Jackson, 2014, p. 6)

As stated by Cioffi-Revilla (2010), the "social sciences or social science disciplines investigate all forms of human and social dynamics and organization at all levels of analysis [...], including cognition, decision making, behavior,

groups, organizations, societies, and the world system" . Social simulation is one of the methods of *computational social science*, the "integrated, interdisciplinary pursuit of social inquiry" (Cioffi-Revilla, 2010, p. 259), along with automated information extraction, social network analysis, social geospatial analysis and complexity modelling. Cioffi-Revilla (2010) provides definitions for these methods.

Axelrod sees (social) simulation as a "third way of doing science. Like deduction, it starts with a set of explicit assumptions. But unlike deduction, it does not prove theorems. Instead, a simulation generates data that can be analyzed inductively. Unlike typical induction, however, the simulated data comes from a rigorously specified set of rules rather than direct measurement of the real world. While induction can be used to find patterns in data, and deduction can be used to find consequences of assumptions, simulation modeling can be used as an aid intuition." (Axelrod, 2003, p. 5)

Hummon and Fararo (1995) see computational methods as a third pillar in addition to the traditional two-component model of science — theory and empirical research (Jackson, 2014, p. 5). While the interplay between theory and empirical research is explanation, data analysis is the interplay between empirical research and computational methods, and simulation is the interplay between theory and computational methods (Hummon and Fararo, 1995, p. 79). Hence, it is often argued that theory drives social simulation. "Most social science research either develops or uses some kind of theory or model" that is generally stated in textual form, sometimes uses mathematical equations, or is expressed in a computer program that can simulate social processes (Gilbert, 2007, p. 1). Although theory-based models have similarities with physics-based models — as the underlying theories can be seen as models of the real world — in the natural sciences "evidence and observation have priority over theory", and "when evidence and theory disagree the theory is changed" (Moss and Edmonds, 2005). Moss and Edmonds argue with regard to social sciences "that evidence should not be ignored without a very, very good reason — including both quantitative and qualitative evidence" (Edmonds, 2015b) when approaching simulation modelling, "especially anecdotal evidence from stakeholders" (Moss and Edmonds, 2005). Likewise, approaches to drive social simulation from empirical research found its way into recent research, as e.g. collected in a special issue in JASSS[13] on "Using Qualitative Evidence to Inform the Specification of Agent-Based Models" (Edmonds, 2015b). The subsection on Evidence-based simulation models goes deeper into the discussion on the topic of evidence-based simulation.

A central term in social simulation is *emergence*:

> "Emergence occurs when interactions among objects at one level give rise to different types of objects at another level. More precisely, a phenomenon is emergent if it requires new categories to

[13]The Journal of Artificial Societies and Social Simulation; `http://jasss.soc.surrey.ac.uk`.

> describe it which are not required to describe the behaviour of the underlying components." (Gilbert and Troitzsch, 2005, p. 11)

Simulation can "give insights into the 'emergence' of macro level phenomena from micro level actions. For example, a simulation of interacting individuals may reveal clear patterns of influence when examined on a societal scale" (Gilbert, 2007, p. 2). Gilbert and Troitzsch (2005) provide detailed background and examples on this matter.

As a specialisation and extension of the simulation types identified by Sokolowski and Banks (2010) and listed above, the following types of social simulation models are of particular importance according to Gilbert and Troitzsch (2005):

- *System dynamics* allows to model a 'target system' — reflecting parts of a real-world system — with its properties and dynamics on a macro level (i.e. as an undifferentiated whole), using a system of (difference or differential) equations. (Gilbert and Troitzsch, 2005, p. 28) Use case are for example research in industrial, economic or political dynamics, such as democracy diffusion (Sandberg, 2011) or economic balance dynamics between supply and demand in housing markets (Özbaş et al., 2014).

- *Microsimulation* is a method to model systems on the microscopic level by focussing on "individual persons with attributes (such as sex, age, martial status, education, employment) and a number of transition probabilities". It is the first approach to add another layer below the macroscopic view of system dynamics. (Gilbert and Troitzsch, 2005, p. 58) Usage scenarios include the prediction of "individual and group effects of aggregate political measures that often apply differently to different persons" (Gilbert and Troitzsch, 2005, p. 57), or the analysis of interdependencies between demographic trends and participation in education (Hannappel, 2015).

- *Queueing models* (or discrete-event models) describe the target system by — typically a set of interacting — entities, which are defined by their attributes, sets, events, activities and delays (Gilbert and Troitzsch, 2005, p. 79) (citing Kheir (1988)). The state of the system changes on the occurrence of events, which gives this model type a peculiar notion of time, as defined in context of simulation paradigms in the beginning of section 2.3. Queueing models are widely used in engineering, workflow management and several other disciplines, as detailed e.g. by Banks et al. (2009).

- *Multilevel simulation models* cater for systems where some parts can be modelled best from one level (say macroscopic), while for other parts a different modelling level (say microscopic) is more appropriate, while these parts are dependent on each other. An example is a "population

with its attributes (for example, its size, its birth and death rates and its gender distribution), homogeneously consisting of a possibly great number of individuals with their own attributes (such as sex, age, political attitudes or annual income)" (Gilbert and Troitzsch, 2005, p. 100). The first simulation environment of this kind was developed by Möhring (1990) (see also Möhring (1995)); another framework is proposed by Camus et al. (2015). Recent research uses multi-level models e.g. for investigating reasons for differences in school effectiveness (Salgado et al., 2014).

- *Cellular automata* model a target system as a spatial representation by means of a regular grid of cells. The cells are attached with a common rule set, and the state of each cell is dependent on the state of the neighbouring cells. In each simulation step, the state is updated for all cells according to the rules. Such models are useful for physical systems (spread of heat in a material), but also for social phenomena like "spread of gossip and the formation of cliques", leading e.g. to "ethnic segregation, relations between political states and attitude change" (Gilbert and Troitzsch, 2005, p. 130). Cellular automata can also be useful for investigating the role of neighbourhood in consumer decision making (Kowalska-Styczeń and Sznajd-Weron, 2016) or for simulating language shifts with the aim to provide forecast whether threatened languages will survive in selected communities (Beltran et al., 2009).

- *Multi-agent models* allow to reflect a target system with high fidelity, while for the above-mentioned types the expected fidelity rather resides on the lower or medium range. The entities making up the real-world model are interacting agents — automata with theoretically unlimited cognitive complexity. The definitions on 'agent-based models' given at the beginning of section 2.3 apply here as well; the following subsection provides additional details on this subject.

In addition, *learning* can be considered in social simulation, either as a independent model type that is based upon artificial intelligence methods like artificial neural networks (Gilbert and Troitzsch, 2005, pp. 217), or by integrating respective features into other model types. The multi-agent model type is a particularly suited candidate for such endeavours, as shown in articles by Conte and Paolucci (2001) or Baptista et al. (2014) and for example captured in the term of "immergence"[14] — modifying the agents' (micro-level) behaviour by "downward causation" from macro-level results (Conte et al., 2013, p. 46).

Complementing the summary above, a quite recent overview on theory, methodology, modelling approaches (including model types) and applications

[14] as counterpart to emergence

is given by Braun and Saam (2014) in their handbook on modelling and simulation in the social sciences[15].

Finally, with respect to the referred publications discussing simulation models, the term *reproducibility* is worth mentioning in this context. Jackson (2014) notes:

> Reproducibility is a "hallmark of the scientific process, since it allows results to be validated independently of the original claimant. Computational models are reproducible in two senses: one, in a very literal sense, the simulation program can be distributed and executed by other people, demonstrating the results directly to them (or not, depending on their independent analysis); and two, in that aspects and elements of the proposed model can be taken out and used in other projects, or modelling ideas from other projects can be combined with or substituted into the proposed model." (Jackson, 2014, p. 11), citing (Buckheit and Donoho, 1995)

2.3.2 Multi-agent models in social sciences

Multi-agent systems came into existence in the last decade of the previous millennium. A definition from the period when agents became state-of-the-art in computer science is given by Wooldridge (2002):

> "The idea of a multiagent system is very simple. An agent is a computer system that is capable of independent action of behalf of its user or owner. [...] A multiagent system is one that consists of a number of agents, which interact with one another, typically by exchanging messages." Agents "require the ability to cooperate, coordinate and negotiate with each other". (Wooldridge, 2002, p. 3)

Wooldridge and Jennings (1995) (with the mentioned cross-references for the first two items) attribute the following characteristics to agents:

- "*autonomy*: agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state (Castelfranchi, 1995)";
- "*social ability*: agents interact with other agents (and possibly humans) via some kind of agent-communication language (Genesereth and Ketchpel, 1994)";
- "*reactivity*: agents perceive their environment (which may be the physical world, a user via a graphical user interface, a collection of other agents, the Internet, or perhaps all of these combined), and respond in a timely fashion to changes that occur in it";

[15] In German language: "Handbuch Modellbildung und Simulation in den Sozialwissenschaften"

- "*pro-activeness*: agents do not simply act in response to their environment, they are able to exhibit goal-directed behaviour by taking the initiative".

These definitions basically apply to any kind of system represented by an agent. Thus, examples of agents can be industrial appliances, that monitor processes with sensors, and react — with some degree of 'intelligence' — to system changes e.g. by varying parameters, thus constituting sophisticated control loops (Leitão et al., 2013).

As mentioned before, the notion of agent in the context of this PhD thesis is directed towards social systems, where agents represent individual human beings, or institutions constituted by human beings. Gilbert (2007) defines agents as follows:

> Agents are "software objects [...], interacting within a virtual environment. The agents are programmed to have a degree of autonomy, to react to and act on their environment and on other agents, and to have goals that they aim to satisfy. In such models, the agents can have a one-to-one correspondence with the individuals (or organisations, or other actors) that exist in the real social world that is being modelled, while the interactions between the agents can likewise correspond to the interactions between the real world actors." (Gilbert, 2007, p. 4)

In social sciences, simulation with multi-agent systems gained importance in the first decade of the current century, coining the term *agent-based modelling and simulation*[16] in conjunction with *individual-based modelling*[17] *(and simulation)* from the (broader) perspective of ecology (Grimm and Railsback, 2005). Individuals can be any kind of living beings including humans, with the distinction that humans have "elements of cognition (e.g. individual reasoning or learning processes)" (Conte et al., 2013, p. 40). Grimm and Railsback (2005) provide a conceptual framework of individual-based models, which also applies to agent-based models. The main concepts are concerned with the following questions:

- *Emergence*: Which "behaviors of the system emerge from adaptive traits of the individuals"? (Grimm and Railsback, 2005, p. 73)
- *Fitness*: How do "individuals estimate the consequences to their fitness that would result from each decision alternative"? (Grimm and Railsback, 2005, p. 84)
- *Prediction*: How do future consequences of decisions affect fitness? (Grimm and Railsback, 2005, p. 91)

[16] ABMS; similar, more commonly used abbreviations are ABM for *agent-based modelling* and ABS for *agent-based simulation*

[17] IBM

- *Interaction*: How do "individuals in an [individual-based model] communicate with, or affect, other individuals"? (Grimm and Railsback, 2005, p. 95)

- *Sensing*: How do "organisms obtain information about their world"? (Grimm and Railsback, 2005, p. 98)

- *Stochasticity*: What "processes in an [individual-based model] should be modeled as stochastic"?(Grimm and Railsback, 2005, p. 101)

- *Collectives*: What are the effects of aggregations formed by organisms on the individual fitness? How differ behaviours and dynamics of the aggregations to those of the individuals? (Grimm and Railsback, 2005, p. 105)

- *Scheduling*: What is the most useful way to represent time? (Grimm and Railsback, 2005, p. 109)

- *Observation*: How can information from the individual-based model be collected in order "to test the model and conduct the analyses the model was build for"? (Grimm and Railsback, 2005, p. 116)

Multi-agent models have in general to deal with a conflict between fidelity (i.e. the complexity of cognition) and scale (i.e. the number of agents that can be mastered by hardware and software used for executing simulations). Hence, except for dedicated high-performance computing models (like e.g. "Virtual-Belgium", Barthelemy and Toint (2015)) and despite dedicated optimisation attempts (Railsback et al., 2017), it is often the case to either find small scale models with rather 'intelligent' agents (as discussed by Bicking et al. (2013) or in Part II of this PhD thesis), or models with a high number of more or less 'dumb' agents (as in the "social force model" by Helbing and Johansson (2007), exploring self-organisation within pedestrian groups, where the "individuals are seen as particles like molecules in liquids or gases, which influence each other through some kind of force"; Lotzmann (2008)).

Multi-agent systems still have relevance nowadays, as some of the referenced examples from above or e.g. the activities in the ESSA[18] community and the articles published in JASSS[19] prove. In computational social science, one of the emerging trends can be seen in the strengthening of data driven modelling and simulation. This subject is discussed in the following subsection on Evidence-based simulation models.

Two kinds of multi-agent simulation models appear in this PhD thesis. Firstly, *policy models* play a minor role mainly for illustrative purposes in Chapter 3. As understood in the context of this PhD thesis, a policy model

[18]The European Social Simulation Association; `http://www.essa.eu.org/`

[19]`http://jasss.soc.surrey.ac.uk`

aims at supporting policy-makers in government and governance by simulating implications and possible future effects of different decision options. Policy models can benefit from features of agent-based simulation, to

- "enable hypothetical what-if questions, by exploring theoretical spaces without precedent" and
- "help communicate sophisticated and complex findings to policy-makers and the general audience", by visualising "the unbiased reasoning with which a model carries out the interactive dynamic that arises from model assumptions". (Hilbert, 2015)

Examples are given e.g. in (Brenner and Werker, 2009) or (Bicking et al., 2013). In a recent contribution, Gilbert et al. (2018) discuss a "selection of examples of computational models used in public policy processes", where the authors "(i) consider the roles of models in policy making, (ii) explore policy making as a type of experimentation in relation to model experiments, and (iii) suggest some key lessons for the effective use of models".

Secondly, a multi-agent model of a criminal network sets the use case example (an thus one of the main contributions) of this thesis. The grounds for this model have been set by Neumann in (Lotzmann and Neumann, 2017) as follows:

> "Epstein's famous postulate 'if you didn't grow it you didn't explain it' (Epstein, 2007) of the programme of a generative social science captures in a nutshell the explanatory account of agent-based social simulation. Agent-based models enable the generation of macro-social patterns through the local interaction of individual agents (Squazzoni et al., 2014). Classical examples include the segregation of residential patterns in the Schelling model, emergence of equilibrium prices (Epstein and Axtell, 1996), or local conformity and global diversity of cultural patterns (Axelrod, 1997). Agent-based modelling has been used for a long time in archaeological research for investigating, for instance, spatial population dynamics ([...] Kohler and Gummerman (2001); Burg et al. (2016)). Likewise there is a growing interest for including culture (Dean et al. (2012); Dignum and Dignum (2013)) and qualitative and textual data in agent-based simulation (e.g. Edmonds (2015b)). Nevertheless agent-based modelling is less used in ethnographic and cultural studies attempting at uncovering hidden meaning of the phenomenology of action. Typically ethnographic research is done by interpretative research using qualitative methods such as thick description (Geertz, 1973) or Grounded Theory (Glaser and Strauss, 1967). The objective of this [modelling and simulation project] is to elaborate a framework for using simulation as a tool in a research process that facilitates interpretation. While applying the

> generative paradigm the objective is growing artificial culture, i.e. an artificial perspective from within the subjective attribution of meaning to social situations. Thereby a methodology will be described for using simulation research as a means for exploring the horizon of a cultural space. [...]
>
> As the project involved stakeholders from the police in a participatory modelling process (Barreteau (2003); Nguyen-Duc and Drogoul (2007); Moellenkamp et al. (2010); Le Page et al. (2015)) for providing virtual experiences to police officers, the objective of the research process is the cross-fertilization of simulation and interpretation [... while doing an] investigation of criminal culture."

Further thoughts on the social-scientific and criminological perspectives are shared in section 2.4.

2.3.3 Evidence-based simulation models

Since the early days of computational social science, actual programming of simulation models for social systems has often been done based on intuition and experience (Lotzmann and Wimmer, 2013a). As further stated by the authors, the way model programmers formalise agent behaviour e.g. by writing rules on the basis of evidence gathered by reading documents or interviewing relevant stakeholders has sometimes the appearance of a "black art" (Edmonds and Wallis, 2002). This approach has proven to be successful for cases when a theory exists between evidence and simulation model (like, for example, shown by Epstein and Axtell (1996)), but also for other cases (such as described by Barthelemy et al. (2001) and Alam et al. (2007)), if the subject of investigation is 'small enough' to allow the modellers to grasp the whole picture of all relevant model aspects.

The situation is different in case of, for example, complex policy models, primarily due to the following facts (Lotzmann and Wimmer, 2013a):

- typically, no (social) theory exists from which such models could be derived;
- instead, such models are constituted by diverse and potentially very large evidence bases;
- as results from such models are intended to have impact on some application domain, the process of validation may have to be extended to a non-scientific community, e.g. to policy makers and other stakeholders.

In the past ten years numerous "research activities have been started with the aim to provide structured approaches for more coherent and, in particular, more provable inclusion of both the evidence base as well as the knowledge and experience of different stakeholders" (Lotzmann and Wimmer, 2013a), such as

in the OCOPOMO project (Wimmer et al. (2012a); see also Chapter 3) or in the (more specialised) context of applying qualitative data for the development of agent rules (Fieldhouse et al. (2016); Edmonds (2015b); Ghorbani et al. (2015); Dilaver (2015); Polhill et al. (2010)). Such research processes can entail "two perspectives, an analysis and a modelling perspective, which are recursively related to each other" (Lotzmann and Neumann, 2017):

- Analysis perspective: The process of recovering information within the empirical data from which certain simulation model elements are derived.

- Modelling perspective: The structured process of the development of a simulation model and the performing of simulation experiments based on the empirical data.

One of the main findings in this context is that *traceability* can be an important enabling factor for evidence analysis and evidence-based simulation model development. In addition to the two perspectives mentioned above, traceability can support the needs of "the stakeholder, who is not directly involved in model development, but who wants 'evidence' in order to gain confidence in model results (and the simulation method as such)" (Lotzmann and Wimmer, 2013a), an issue raised by Waldherr and Wijermans (2013).

This concept of traceability is related to the one introduced in the context of software engineering in the subsection on Traceability, in a way to keep track of empirical foundations provided by stakeholders, such as scenarios or police interrogations protocols, together with background information. Thereby the provenance[20] of arguments by stakeholders is ensured "through the establishment of traces and links between sources of information [...] provided by the stakeholders [...], and the simulation models developed by [modelling] experts. The links show the evolution of formal elements of a simulation model from the description of the real-world [...] subject of the model" (Lotzmann and Wimmer, 2013a).

For realising this kind of traceability, the approach of conceptual modelling plays a key role — and to an extent takes the place of theory. Conceptual models can act as a hub of trace information, which facilitates "navigating from simulation outcomes back to the simulation model back to the conceptual model, and finally back to the [evidence] documents" (Lotzmann and Wimmer, 2013a).

A recent (November 2018) literature search shows that the benefits of traceability start to be recognised by the agent-based social simulation community, yet other realisations than the ones mentioned above are quite scarce up until now.

[20]In the Oxford English Dictionary, provenance is defined as "the origin, or the source of something, or the history of the ownership or location of an object, especially when documented or authenticated". For a more detailed elaboration of definitions of 'provenance', see Munroe et al. (2006).

2.3.4 Simulation and Software Engineering

The aspects of software engineering and implementation compiled in section 2.2 basically also apply to the domain of computer simulation. However, Jackson (2014) points to the issue that "despite many varieties of development frameworks available today, they all assume a production model at their core: a product is developed for a client. There is a gap between current methods of software development and the needs of the research environment" (Jackson, 2014, p. 8), citing (Kelly, 2007). Consequently, there are several examples of research to fill in this gap, e.g. by Bommel et al. (2014) or Klügl (2010) and Klügl (2013).

In this view, some of the software engineering aspects rely on particular conceptions in the context of simulation. Validation is a prominent example of such an aspect. Both validation and verification "are essential prerequisites to the credible and reliable use of a model and its results" (Sokolowski and Banks, 2009, p. 121). While the process of verification is quite similar to the definition in software engineering, validation encompasses further facets. "The process of validation assesses the accuracy of the models", i.e. the degree to which the real-world system of interest is represented in the simulation model and reflected in the simulation results (Sokolowski and Banks, 2009, p. 126). According to (Troitzsch (2004), citing Zeigler (1984)), validation of a simulation model should not only regard the observed (replicative validity) and unseen behaviour (predictive validity) of the real-world system, but also structural validity, i.e. if the simulation model "mimics in step-by-step, component-by-component fashion the way in which the [real-world] system does its transactions" (Zeigler et al., 2000, p. 31).

Other aspects regard input and output data of simulation models. Simulation as a research method involves experimentation with simulation models (Gilbert and Troitzsch, 2005, p. 14). Parameters of the experiments are inputs of the simulation model. These can originate directly from user inputs (similar to control inputs from the user of a flight simulator) or from stochastic processes (Sokolowski and Banks, 2009, p. 92), but often arise from a careful experiment design (see e.g. Barton (2010) or Reinhardt et al. (2018)). In the course of a simulation experiment, many simulation runs are executed, with some parameters kept static and others varied over specific ranges (Sokolowski and Banks, 2009, p. 92). Values and ranges for parameters can be elicited by sensitivity analyses (Chattoe et al., 2000), informed from observation of the real-world system (Gilbert and Troitzsch, 2005, p. 19) or external sources of empirical date (such as statistics) (Gilbert and Troitzsch, 2005, p. 63), but also the same sources and methods applied for requirement analysis in software engineering can generate inputs for simulation experiments. In particular the data analysis and participation approaches outlined in subsections Requirement Analysis and Participation are relevant in the context of this PhD thesis. Hence, requirements can be reflected in both the software specification and in the input data for the resulting computer program.

Special attention has to be given to the output data of simulation models, too. Simulation runs generate data with numerical (e.g. changes of an output variable over time) or textual content (e.g. logs of occurring events), typically in a structured form with meta data (like timestamp and originating component). Such data can be simply presented in tables or text files, but in many cases visually processed representations are more effective. "Visual representations are reshaping the exploration, analysis, and dissemination of simulation data, and promise to revolutionize knowledge discovery and innovation" (Sokolowski and Banks, 2009, p. 103). A selection of relevant visualisation methods is presented in section 7.2.2. Beyond that, frequently used visualisations are compiled in the following list, together with references to usage example:

- Diagrams representing numerical data, such as simple time lines, but also charts that allow aggregated or multivariate visualisations, such as e.g. box plots or scatter charts; see e.g. (Troitzsch, 2016a) or (Alves Furtado and Eberhardt, 2016).
- Network of components (e.g. agents); see e.g. (Alam et al., 2007) or (Lotzmann and Neumann, 2017).
- Diagrams for spatial visualisation (e.g. for traffic simulation), such as regular grids, Voronoi diagram or maps based on GIS[21] data; see e.g. (Le Page et al., 2015), (Polhill et al., 2010) or (Barthelemy and Toint, 2015).
- 3D-visualisations and virtual reality; see e.g. Crooks et al. (2009).

Most of these diagrams may be attached with additional information (forming 'rich pictures' Avison et al. (1992); see e.g. Figure 7.13), or appear in animations, where typically changes over time are visualised as an additional dimension to the displayed content (e.g. in traffic simulation and typical Netlogo[22] models). A more general overview on approaches in analysing and reporting outputs of agent-based models provide Lee et al. (2015). An even broader context by reviewing contemporary simulation tool is given by Abar et al. (2017).

2.4 Application in Criminology and Social Sciences

This final section gives the context of the application field of criminology relevant to the use case example presented in Part II, in relation to simulation and combined with software engineering.

[21] Geographical Information Systems (Raper, 1993)
[22] `http://ccl.northwestern.edu/netlogo/`

Information and communication technologies have been widely used in criminology (the scientific perspective) and criminalistics (the practitioners' perspective). Some of the main applications regard the collection, (shared) storage and analysis of data from and about criminals or suspects. With the apparently increasing threat by terrorist attacks and 'cybercrime' — criminal activities involving the World Wide Web and other contemporary communication technologies — the incentive is high to massively strengthen the respective technological capabilities of police forces. Collections of topics and examples are given e.g. by Antinori (2008) or Byrne and Marx (2011).

Also, simulation plays a significant role in both criminology and criminalistics. Research in criminology is looking, among others, at the inside perspective of police, or at the victims' perspective. An example of the former is a "multi-agent architecture for regulated information exchange of crime investigation data between police forces" by Dijkstra et al. (2005); the latter is exemplified in a simulation of "people's reaction to norm violation" (the "bystander effect") by Gerritsen (2015). Research directed towards criminalistic exploitation often regards prediction of crime occurrence, such as shown by Kang and Kang (2017), Malleson et al. (2013) and Bosse and Gerritsen (2008), or by Groff et al. (2018) who give an overview on potential contributions of agent-based modelling.

An important subject of investigation is extortion racketeering, a specific form of criminal dynamics usually found in organised crime, with Mafia as the most prominent example (but also common in any kind of criminal network). How agent-based simulation can be applied to this topic is shown by e.g. Elsenbroich (2017), Nardin et al. (2016b) and (Troitzsch, 2016b). While these articles focus on the relation between criminal and legal world, Neumann et al. (2017) concentrate on the different perspective of internal conflicts within the Mafia.

This inside perspective of criminal organisations or networks is often addressed with the goal to enlighten such covert structures of society. Data analysis methods such as social network analysis (SNA; Otte and Rousseau (2002)) or text mining as well as qualitative research can be useful in this regard, as shown by Neumann and Sartor (2016) or Bright et al. (2012). The additional step to feed such information into simulation models is taken by the research described in Part II of this PhD thesis, with the aim to address research questions from both criminology and social science.

The criminology perspective is outlined in (Van Putten and Neumann, 2018), while Neumann and Lotzmann (2016a), Neumann and Lotzmann (2016b), Neumann and Lotzmann (2016c) and Lotzmann and Neumann (2017) focus on the social-scientific questions of criminal culture and the relation to social norms. Neumann defines the area of interest as follows:

> The use case example is dedicated to "the investigation of criminal culture. While ethnographic research goes back to classical

> studies of e.g. Malinowski or Margaret Mead on tribes in the Pacific islands (Malinowski (1922); Mead (1928)) it has expanded to studying cultures of various kinds such as youth (Bennett, 2000) or business culture (Isabella, 1990). Thus it is reasonable to use the field of the criminal world for interpretative investigations. The objective of the research, imposed by the stakeholder interests, was to investigate a specific element of criminal culture, namely modes of conflict regulation in the absence of a state monopoly of violence. For this purpose norms and codes of conduct of the 'underworld' needed to be dissected. While individual offenders might pursue their own path of action, as soon as crime is undertaken collectively, i.e. in the domain of organized crime, the necessity for standards that regulate interactions emerges in the criminal world as it does in the legal society. These can be described as social norms (Gibbs (1965); Interis (2011)). Thus studying norms and codes of conduct in the criminal world provides a perfect example for studying the emergence of a specific element of culture, namely social norms." (Lotzmann and Neumann, 2017)

In criminal networks trust plays a central role to maintain intra-organisational stability. Colquitt et al. (2012) define trust as "confident, positive expectations about the words, actions, and decisions of another in situations entailing risk". As Van Putten and Neumann (2018) point out, "a criminal organization needs to rely on the commitment of the members to the organization. For this reason, trust is essential" (Van Putten and Neumann, 2018, p. 6). Absence of trust triggers violence, which can cause the breakdown of such networks. Trust in criminal organisations is needed, since formal means for conflict regulation as established in the legal world ('state monopoly on the use of force') are not available in the criminal world.

Part I

Modelling process

Chapter 3

Concept and tools for evidence-based policy modelling

3.1 Introduction

This chapter sets the general frame of the research presented in this PhD thesis in multiple respects. Firstly, the basic modelling process design is outlined to be adapted (in Chapter 4) and applied in the use case example (presented in Part II). Secondly, the individual steps of this process are sketched from the conceptual as well as from the technical side, together with a presentation of the associated software tools.

The contents presented in this chapter are the results of cooperative work performed in a research project (OCOPOMO) that aimed to tackle specific problems discussed in section 2.3 by applying methods introduced in section 2.2. The author of this thesis participated in this research project primarily as simulation software and model developer. His main contributions in this project are basis of Part III of this thesis, and have partly been previously disseminated in (Lotzmann and Wimmer, 2012), (Lotzmann and Wimmer, 2013b), (Lotzmann and Wimmer, 2013a) and project deliverables.

The chapter is structured as follows: First, the OCOPOMO research project is introduced with its assumptions, goals and outcomes, together with references to the relevant publications and project deliverables. Next, the modelling process for evidence-driven policy simulation as theoretical key result of the project is described, providing details and examples for each of the process phases: domain-knowledge collection, conceptual modelling, model transformation, simulation modelling and simulation result generation and analysis. Finally, the traceability concept interconnecting the process phases is defined.

3.2 The OCOPOMO project

OCOPOMO[23] — the acronym for Open Collaboration for Policy Modelling — was a research project which started in February 2010 and ended in May 2013. It was co-funded by the European Commission in the Framework Programme 7 with a theme in ICT for Governance and Policy Modelling, and brought together seven partners from universities, public administration and the private sector, situated in Germany[24], Italy, Poland, Slovakia and the UK. This project provided the context in which considerable parts of the ideas and work presented in this PhD thesis have been developed. A number of public deliverables are available; in particular relevant for the aspects targeted in this chapter are (Furdik et al., 2013), (Bicking et al., 2013) and (Scherer et al., 2013a). What this project made particularly ambitious was that on the one hand it brought together a broad range of people from Information Systems Research, E-Government, Computer Science, Public Administration and Private Consultancy, on the other hand the goal to deliver an entire package of means for enhancing electronic support for policy making, starting with a process description for an integrated participatory policy formation, a software toolbox supporting the phases of the process, right up to elaborated guidelines for applying both the process and toolbox in different applications. By integrating experiences from the various research fields and practice, in all of the developed artefacts both approved concepts from the specific fields as well as new approaches or methods have been introduced.

The research done within the OCOPOMO project was based upon the assumption that simulation models derived from social theories have not proven to be successful for practical applications supporting decision processes of policy makers, i.e. "to forecast correctly the social impacts of policy initiatives and implementations" (Moss, 2011). Models based on "observation and evidence collected independently of the models" (Moss, 2011) — as studied in OCOPOMO — are an alternative for theory-driven models. The evidence should be comprised not only of statistics, reports and other available background material, but — most importantly — of experiences and opinions of directly involved stakeholders which have to be documented, in this case in different kinds of scenarios[25].

As a reusable conceptual framework for dealing with evidence-based model development, an adequate process definition had to be developed (Scherer et al., 2015). As identified in the requirements analysis during the project, two objectives are of particular relevance: On the one hand, this process must support the management of evidences, i.e. it must regard how to keep links from any element within any artefact throughout the model development activities

[23] `http://www.ocopomo.eu`

[24] with the University of Koblenz-Landau, Germany, as the coordinator of the consortium

[25] In this context 'scenario' is referred to the scenario method as introduced by Carroll (1995). A definition is given in section 2.2.2.

to the original documents. On the other hand it should help to structure the modelling process in order to elaborate 'best practices' for certain procedures within the overall modelling task. One of the most important added value that can be expected from applying such a process is a means for creating meaningful and usable documentation of the model for all involved persons and roles. Additionally, the numerous drawbacks of 'modelling directly to program code' can be avoided in most cases (Grimm and Railsback, 2005, e.g. p. 306).

More generally, applying procedures for creating simulation models which fulfil certain quality standards — adopted from software engineering — can be regarded crucial to ensure acknowledgement and, hence, prospective success of the simulation method in application areas like government and management. The approach to solve problems is in large parts similar to approaches for developing e.g. large customer-specific software systems: From the detection and avoidance of misunderstandings due to special terminologies in the requirements analysis phase, to allowing the contractees or stakeholders to accompany the development activities, right up to providing means for testing and understanding the final product or other results. The challenge in terms of software development is to introduce means and procedures to structure the development process adequately. The whole discipline of software engineering deals with these issues, and the findings made there have sustainably revolutionised software development and the entire field of computer science. It seems natural to rely on these findings also for creating simulation models, since large parts of work necessary in this context is very similar to such for software development. This is subject of numerous research activities during the last decade (e.g. Siebers and Davidsson (2015) give an overview, see also section 2.2), and a lot of achievements have already been made. But there are also considerably many differences, which make it difficult to apply approaches like model-driven software development to simulation model development.

As introduced by Wimmer (2011) and also addressed e.g. in (Wimmer et al., 2012b) and (Lotzmann and Wimmer, 2013a), in OCOPOMO a contribution was conceptualised and implemented in order to relieve these difficulties by integrating software engineering with participatory policy modelling as a subfield within the 'traditional' experimental social sciences. This was achieved by a novel approach for engaging stakeholders in policy development. Stakeholders were collaboratively involved in the development of scenario texts relevant in the context of a policy under discussion (Scherer et al., 2015). In this regard the term policy is referred to strategic areas of complex decision-making with various stakeholders having conceivably diverging interests. In OCOPOMO, public policies have been investigated and modelled such as renewable energy policy of the Kosice region in Slovakia, housing policy of the city of London or the distribution of structural funds in the Campania region in Italy (details on these pilot use cases can be found in (Bicking et al., 2013)).

The following sections provide an overview on the development process defined in OCOPOMO as originally published in (Wimmer et al., 2012a) and

(Scherer et al., 2015). Here the focus should be on the illustration of the individual process phases together with the developed software-tool support.

3.3 The OCOPOMO process

The overall OCOPOMO policy model development process as conceptualised by Wimmer (2011) consists of six phases:

1. An initial scenario is developed by policy makers or domain experts, describing a policy prospect.
2. Stakeholders are involved to generate scenarios of potential further aspects of the policy on the basis of the initial scenario. These are complemented with background documents to provide evidence for statements in the scenarios.
3. The policy case is then conceptualised and
4. implemented in simulation models by modelling experts.
5. Simulations are run to generate outcomes.
6. The results of the simulations are exposed to the stakeholders. They compare their scenarios (of phase 2) and the simulation outcomes in order to either update their scenarios (and start another cycle with phase 3), or endorse the simulation results, i.e. agree that these are consistent with the inputs provided in phase 2.

Within the process, four major thematically distinct subject areas can be identified:

- *participation* — the definition of a role for accessing domain knowledge and stakeholder opinions;
- *collaboration and scenario editing* — the technical means for enabling participation;
- *simulation modelling* — the method applied for enhancing knowledge in the domain;
- *conceptual modelling* — in a sense the heart of the process as it serves as a connecting link — both from a content-related and a technical perspective — between the other subject areas.

Figure 3.1 shows a visualisation of the process steps in relation to each other and to the related subject areas. The artefacts exchanged between the process steps are also highlighted in order to allow for a more practically oriented

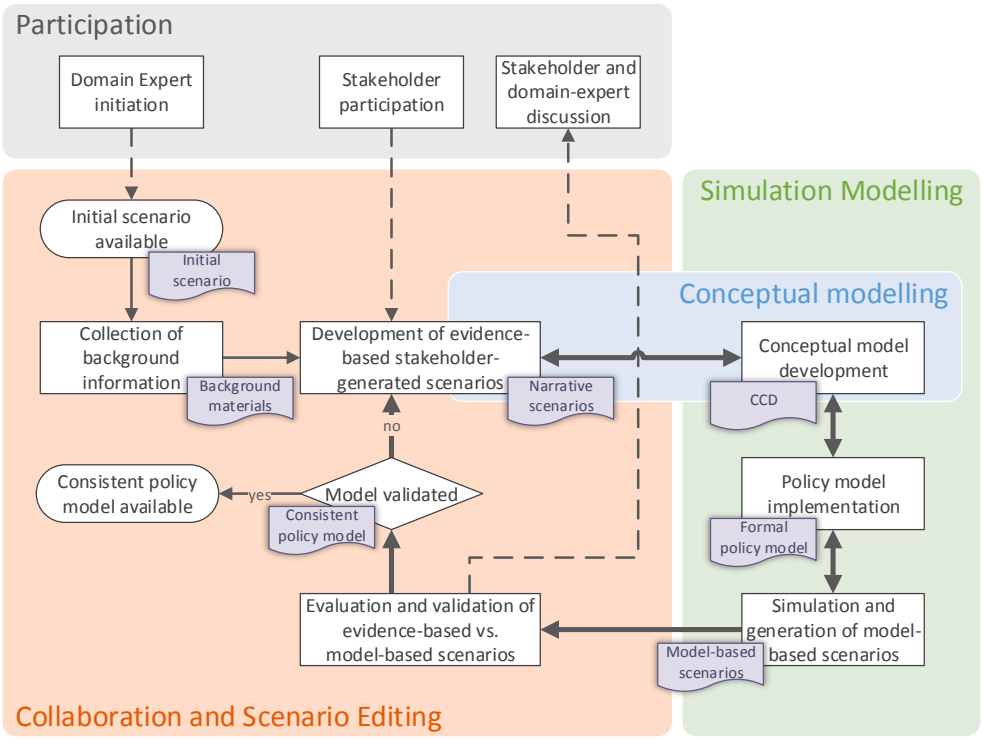

Figure 3.1: Overview of the evidence-based modelling process developed in OCOPOMO (based on Scherer et al. (2015)). Background boxes represent subject areas of the process; the flowchart highlights the activities, with solid lines showing the process flow and dashed lines depicting influential relationships.

description. The process starts with an initiative from the side of the domain experts, which in a sense have the role of 'orderer' (or 'customer') of the modelling endeavour. They specify the case to model as an initial scenario from their perspective, typically in an abstract and generic way. This scenario is then augmented with details and background information available from various (public) resources. This task is performed by analysts and modelling experts, who, in turn, can be regarded to have the role of 'contractors'. This process step relies on collaboration between the domain and modelling experts.

This step is the prerequisite for the following phase of collaboration, in which potentially any kind of stakeholders contribute with their knowledge and opinions. The stakeholders can again be domain experts in areas which are concerned by or concerning the topic of modelling, but also the general public may be included. The result of this process step are narrative scenarios, every single one of them expressing the perspective of a specific stakeholder or

stakeholder group. Additionally, the stakeholders are asked to contribute with further background material, in order to achieve a profound and comprehensive collection of information describing the modelling topic.

All the available information — the initial and stakeholder scenarios and background material — then become the basis for designing a conceptual model, bringing all the information together in a structured and semi-formal way. This task is performed by modelling experts and results in the creation of a consistent conceptual description (CCD). During the development of the conceptual model, gaps or ambiguities in the material may occur. These have to be resolved with a step back into the collaboration phase, where the domain expert and stakeholders are requested to provide additional or more specific information.

Once the conceptual model has reached a state mature enough to inform a formal simulation model, the implementation task can start. This task is typically performed in an incremental way, as the path to formalisation usually requires additional and very specific details. Such details can either be requested from the stakeholders, or can be filled with assumptions by the modellers, preferably gathered by profound desk research.

As part of the incremental model implementation, executing already early version of the simulation model might — besides the identification and correction of software errors — also lead to the discovery of new gaps and ambiguities in the conceptual model. Hence, the option to further adapt or extend the conceptual model or to request additional information from previous process steps need to be considered. The final result of modelling activities are one or more final simulation models, with which 'productive' simulation runs are performed, which generate different sorts of outcomes. These outcomes are compiled and elaborated into model-based scenarios, i.e. textual descriptions of possible courses of events, addressing the original questions of the policy expert 'customer'.

These model-based scenarios have then to be presented to the policy experts and stakeholders in order to be evaluated by them. Main objective of this evaluation and discussion is to make sure that the relevant questions and aspects are addressed by the model or can be easily be included in the model, e.g. via parameter variation. If this is the case, then a consistent policy model has been created. In many cases it can be expected that the simulation results provoke questions reflecting completely different views on the topic (which, in fact, is a sign that the work was done properly and the model has achieved its goal to enhance knowledge), which might then lead to a next iteration of the modelling process. This aspect of having several model iterations is crucial especially for the case where future processes are in focus, i.e. where the purpose of the model is more in the direction of revealing possible alternatives for given initial policy constellations.

In the project, an integrated ICT toolbox (Furdik et al., 2013) has been created to support the policy development process and to ensure a smooth

transformation of policy inputs by stakeholders into the formal policy models. Figure 3.2 gives an overview on the architecture. The ICT toolbox consists of tools related to the building blocks and the interfaces between them, in detail:

- A participation platform[26] that enables stakeholders to collaboratively develop their scenarios, to upload and share background documents, and to discuss among each other about views and issues of a policy (supporting process phases 1 and 2), and to present simulation results and traces to the stakeholders. A more detailed characterisation is given in section 3.3.1.

- A Consistent Conceptual Description (CCD) tool[27], which enables policy modellers to develop a conceptual model of a policy domain. The CCD tool supports annotation of scenarios and background documents and therewith keeps track of provenance (supporting process phase 3). Section 3.3.2 gives a brief introduction to this tool, which will then be addressed again in the application example in section 5.4.

- A simulation environment (Declarative Rule-based Agent Modelling System, DRAMS)[28], which supports policy modellers in programming, running and analysing policy simulation models (supporting process phases 4 and 5). This tool is introduced in section 3.3.4. As major contribution of this PhD thesis, the usage of the tool is elaborated in the context of the application example in Chapter 6, while a detailed technical perspective is provided in Part III of the PhD thesis.

- A CCD2DRAMS transformation tool[29], which supports the semi-automatic transformation of conceptual policy model constructs into code of formal policy models (link between process phases 3 and 4). Section 3.3.3 gives more information on this tool.

The architecture of the toolbox is characterised by two sub-systems:

- A web-based application based on the Alfresco CMS[30], integrating components for collaboration and content management. Typical users are policy experts, stakeholders and facilitators, with the latter chairing the collaboration activities.

[26]Responsible project partners were the Technical University of Košice and InterSoft a. s., both Slovakia (Butka et al., 2011).

[27]Responsible project partner was the research group eGovernment of the University of Koblenz, Germany (Scherer et al., 2015).

[28]Responsible project partners were the research group eGovernment of the University of Koblenz, Germany, and Scott Moss Associates as well as the Centre for Policy Modelling (CPM) of the Manchester Metropolitan University, both UK (Lotzmann and Meyer, 2011b).

[29]Responsible project partner was the research group eGovernment of the University of Koblenz, Germany (Lilge, 2012).

[30]`http://www.alfresco.com/`

- An Eclipse modelling platform, and integrated development environment with modelling-related components, implemented as Eclipse features. Typical users are modeller, programmers and simulation analysts.

Each sub-system is built around content repositories which store the artefacts handled by the (local) components, but also serve as bases for communication between the two sub-systems. Therefore, on the Eclipse side a data repository client provides functionality to access the Alfresco data repository for exchange of different kinds of artefacts. These artefacts are initial and stakeholder scenarios together with background documents as input for the modelling platform, and simulation results with traceability information as content to be shown and discussed in the collaboration platform. Results of this discussion, on the other hand, might again be input for further modelling activities.

More detailed descriptions of the OCOPOMO policy development process phases and the respective tools are given in the following subsections. Aim of these parts is to provide a link between the process and the toolbox. In each of these subsections, the functional spectrum of one dedicated toolbox component is described and the related process steps are highlighted. These sections are intended to address issues relevant to particular readerships (in order to provide a proper foundation to the subsequent chapters of this PhD thesis): In the first place the modeller perspective is emphasised, i.e. for the collaboration part less details are provided. In the second place the more general user perspective is taken into account, i.e. only the concepts these tools are based upon are outlined and their usage is demonstrated with simple examples.

3.3.1 Domain-knowledge collection

This section comprises the phases 'Collection of background information' and 'Development of evidence-based stakeholder-generated scenarios'.

Starting point for a policy modelling initiative is usually a description of a policy case for which preconditions, constrains, consequences and effects, risks and side effects and — to put it briefly — the prospects for success or failure should be investigated. This description of the policy case is supposed to be available as a natural-language description, together with supporting information from reports or statistical figures and opinions of affected stakeholders. The endeavour now is to put these texts and figures together into a simulation model in a way that the model abstractions introduced by the modelling experts meet the needs and expectations of the domain experts and also of the stakeholders. All these materials constitute what is called the evidence base of the model. With the simulation it must be possible to trace the relation between the evidence and the details of the simulation model as well as the resulting data from simulation runs. A prerequisite for the success of such a modelling initiative is to gather and store the documents of the evidence base in a well-structured way.

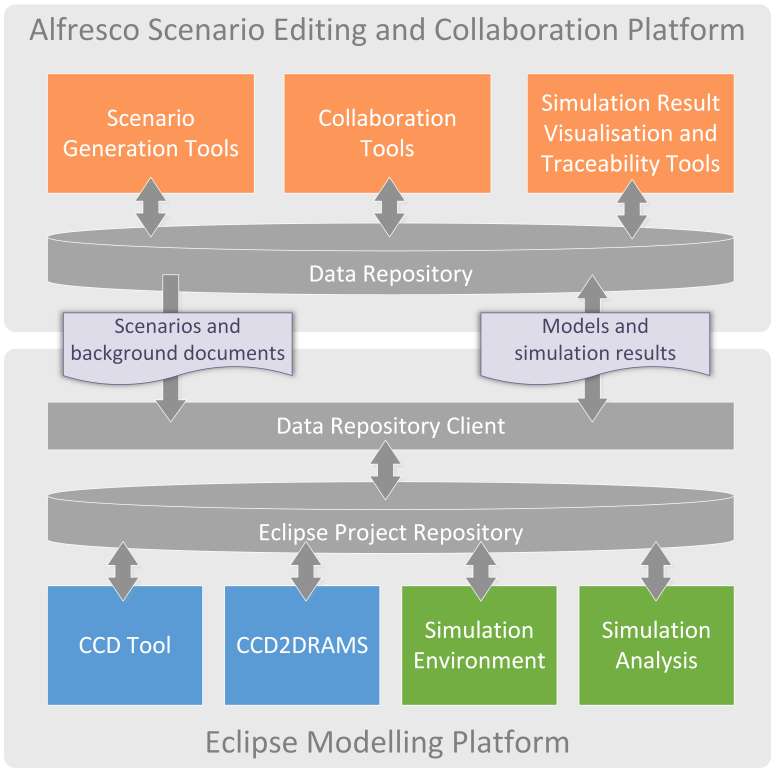

Figure 3.2: OCOPOMO toolbox architecture overview (following Furdik et al. (2013))

Source documents for modelling

Source documents for the envisaged modelling tasks are different kinds of scenarios, outcomes of scenario studies as defined in section 2.2.1.

An initial scenario regarding a policy case to model is provided by 'policy owners', a group among the domain experts with the interest to start a discussion on this policy case (Wimmer and Bicking, 2011). The initial scenario can be a description of the current state of some 'world properties', or it can be a story of possible and/or intended future states of the 'world'[31]. Both kinds can serve as starting point for modelling, with different objectives: While for the former the aspect of understanding the coherences of factors that matter the current state by simulation is in the foreground, for the latter the exploration of possible future states (in a sense of forecast) is the aim. In any case, the initial scenario must be supported by background documents, providing facts and further details which are omitted in the scenario, but which are necessary to understand the matter of the policy case. Background documents can be any kind of written information, statistical numbers or figures. Sources can

[31] 'World' is referred in this context to a system as part of the real world — like a state — that can be influenced in some ways by a body — like a government can do by political decisions and policies.

be e.g. reports or statistics from administrative bodies, newspaper articles or web sites — any type of information that can be stored as a digital file.

The initial scenario and background documents are not only the basis documents for modelling activities, but also for discussions on the included topics with other affected persons, the stakeholders. These discussions are moderated and documented by facilitators, and performed by applying an open collaboration approach. In view of the prospective electronic processing of the information, it is favourable to realise the platform for discussions as a web-based portal which provides collaboration tools with the facility to comment on the documents in the repository — like a Wiki embedded is a content management system. Goal of this interaction between stakeholders, facilitators and policy experts is the provision of 'stakeholder scenarios'. In many cases these provide more realistic and accurate views on the relevant topics, and are suitable to capture a much broader range of opinions and experiences than it would be possible to be done by a small group of experts. Hence, capturing these different perspectives constitutes the strength of participatory modelling.

Tool Support

In order to manage the scenarios together with the possibly large number of supporting documents, an appropriate ICT infrastructure in form of a data repository is required as backbone for the integrated toolbox (Butka et al., 2011). The repository gives access to the various scenario editing and collaboration components, but also to the Eclipse-based (off-line) modelling components, as mentioned above and shown in Figure 3.2.

For collaborative generation of the stakeholder scenarios, a basic requirement of the ICT infrastructure is the integration of collaboration tools with text editing facilities for writing texts and discussing arbitrary documents online. A separate part of the repository is then to be reserved for consolidated scenarios and other documents, which will be used for the next phase of the policy modelling process, the development of the conceptual model.

These requirements have to be regarded for setting up an appropriate software solution, taking existing content management systems (CMS) into account. In addition to the premise to be open source software, further selection criteria are typically relevant: On the one hand the 'on-board' availability of as much as possible of the needed functionality, on the other hand the flexibility in respect to providing an API for implementation and integration of any missing functionality. As mentioned above, the CMS selected in OCOPOMO was the Alfresco platform. A discussion on reasons for selecting this particular platform together with some architectural and technical details on the system and the performed adoptions is given in (Butka et al., 2011). Table 3.1 summarises an overview on the included components and their functionality.

The screen-shot of a web browser in Figure 3.3 shows the discussion page for a scenario for one of OCOPOMO's use cases that served as pilot in the project. This particular pilot aimed in investigating different policy strategies

Component	Module	Functionality
Scenario Generation Tools	Wiki	online-editing of scenarios and other texts
	Document Management	access to data repository
Collaboration Tools	Dashboard	personalised view to data repository and functions
	Discussion	add comments to texts and other content
	Chat	chat room
	Polling	setting up and participating in polls
	Calendar	calendar with relevant events
Simulation Result Visualisation and Traceability Tools	Annotations	traceability information (links and short descriptions)
	CCD Viewer	applet that enables browsing of the various CCD diagrams

Table 3.1: Functionality provided by Alfresco collaboration platform

for reducing heating energy consumptions and, thus, costs for citizens of the region of Košice in Eastern Slovakia. The other examples shown subsequently in this section will also embark on material from this pilot. Further information is available in (Bicking et al., 2013). The actual scenario text (here in a non-editable format) is presented on the left, while on the right hand side available functions can be accessed. Not visible in the screen-shot is a discussion area located below the text.

3.3.2 Conceptual modelling

Having the information for the subject to be modelled collected, the process of formalisation can begin. Basically this could be done in a mental effort by an individual modelling expert or a small team of experts, who have the formal system aspects already in mind during the data collection phase (where they participate as drivers or participants). As already mentioned, this is a common approach in the field of social simulation and certainly also in other disciplines, and is beneficial when rapid development of prototypes with restricted complexity is desired. The dependency on personal skills as shaping (and also restricting) factor of the complexity and quality of the model formalisation, and the lack of transparency on decisions about formalisations and abstractions can have unfavourable effects. In particular the latter argument is likely to diminish the benefits of an elaborated data collection as model

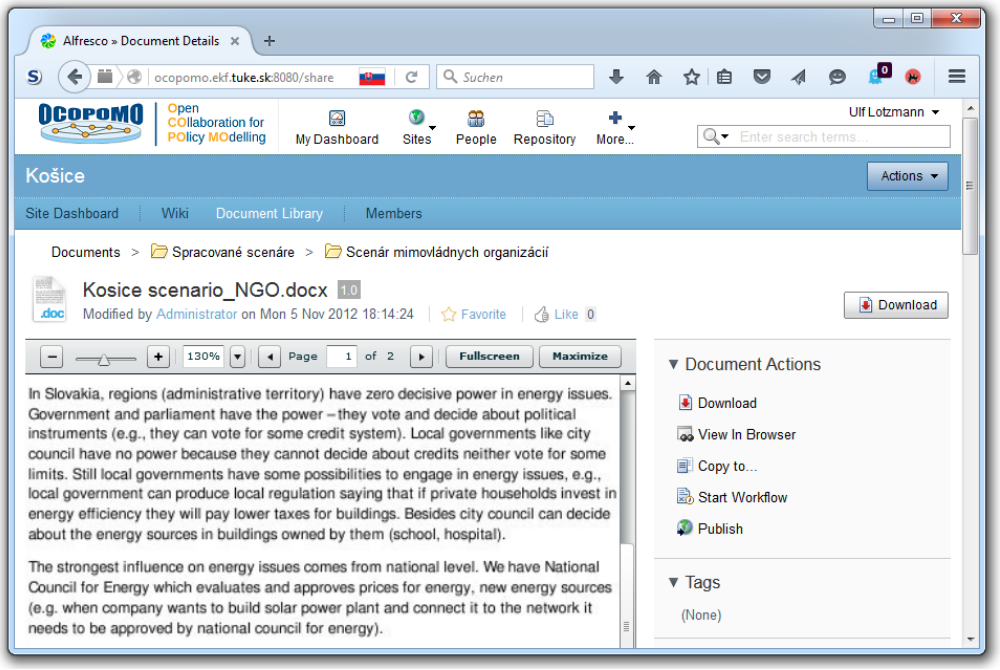

Figure 3.3: Example for stakeholder scenario, displayed in the Alfresco collaboration space

foundation, as there will usually be no traceable documentation for the single formalisations which links to the evidence these decisions are based upon.

For this reason, and also for the sake of reducing the complexity of the formalisation task by splitting the problem into smaller and better manageable sub-problems, a distinct phase dedicated for conceptualisation of the modelling task is essential. Here the method of conceptual modelling as applied in software engineering and defined in section 2.2.3 comes into play. The following subsections show an approved solution for this 'Conceptual model development' phase.

The CCD concept and design

A conceptualisation of a policy model must fulfil two key requirements:

On the one hand, it should present the subject of the model both as succinct and as comprehensive as possible, and in a way such that stakeholders and other non-modelling experts are not left behind when it comes to understanding the abstraction and formalisation the model is intended to be. I.e., also for non-modellers, it must remain comprehensible whether the key elements of the modelled system are present and linked with the other elements of the model in a correct and sensible way. This includes the functionality to keep track of the links between inputs from the data collection phase and the

elements of the conceptual model.

On the other hand, it should be closely related to the means of formalisation with which the actual simulation models are to be realised. This means the conceptual model must provide sufficient information on how to program the elements of the simulation model, and how these elements interact among each other. In cases where the conceptual model alone does not provide enough details, the link to relevant resources (the evidence base) must be made easily available.

These two requirements pose noticeable deviations from the approach to conceptual modelling established in software engineering, so that these methods are not immediately suitable for the purpose of policy modelling. Due to the complexity of the subject to model — which per definition includes models of human behaviour — requirements for the formal model are not clear but rather vague at the beginning, often changing in the process of modelling and even when experimenting with the formal model. Furthermore, stakeholders and field experts (e.g. social scientists) should be deeply involved at the stages of formalisation, but often feel overwhelmed by the imposed technical view on the topic which software engineering methods entail. Hence, the aim is to define a conceptual modelling approach that satisfies these particular needs.

In more technical terms, and with respect to maximising the utility of conceptual modelling, the following main requirements can be formulated (partly summarising requirements defined by Scherer et al. (2013b)):

- The conceptual model must be able to expose model details — like the structure and dependencies of modelled entities and the logic behind the model dynamics — to the stakeholders and other interested audience beyond the usual textual documentation, but without the necessity to understand program code.

- The language of the conceptual model must be rich enough to specify a degree of formalisation to allow automatic generation for at least parts of the simulation model code, but at the same time general enough to be compatible with a variety of specific domain languages and even natural language, in order to be understandable by stakeholders and non-modelling experts.

- The conceptual model must be independent from any programming language or simulation platform, but allow to automatically generate model code in different programming languages by employing dedicated transformation mechanisms.

- The conceptual model must provide means for documenting the model as well as the modelling process, with the functionality to maintain traceability, i.e. connecting links between the data collection (i.e. the evidence base) and the formal simulation model.

A solution to cope with these partially conflicting requirements (broadly speaking: complete formal specification versus readability by non-modelling experts) needs to be a sensible compromise. In OCOPOMO, this was achieved with the Consistent Conceptual Description (CCD), published (among others) in (Scherer et al., 2015) and (Scherer et al., 2013b). It embarks on concepts from agent-based social simulation (or, more generally, microscopic or individual-based modelling approaches), but is not necessarily restricted to this domain (Neumann and Lotzmann, 2016a), and beyond that independent from specific domain and formal programming languages.

To give a basic understanding of this solution, Figure 3.4 shows a simplified meta-model of a CCD (following (Scherer et al., 2013b) and (Lilge, 2012)), i.e. a definition of the language with which the conceptual models are formulated. There are three parts of the meta-model highlighted, each representing a specific concern of the conceptual model:

- The ontology (see section 2.2.3) specifies the parts of which the structure of the model is comprised. These are the concepts (or classes) of possible elements of the system the model describes.

- The facts then are the actual entities (or objects) of the concepts specified in the ontology.

- The behaviour finally specifies the dynamic aspects of the modelled system, i.e. how the facts change due to specific events or in the course of time.

Upon closer inspection of the meta-model, the root concept *CCD* is composed of four concept classes, each specifying a type of entity of the conceptual model: *Actor* and *Object* forming the static structure in the ontology of the CCD, while composites to *Action* and *Condition* classes are linking to behavioural aspects.

Within the ontology, an arbitrary number of actor and object types can be contained. Actors refer to (pro- and re-) active entities (like e.g. humans); objects are concepts for passive elements, which can be of physical, mental or any other nature imaginable, and are typically possessed or treated by actors. Both actors and objects can have subtypes as a means for specifying a hierarchical specialisation (e.g. an object of type 'furnace for domestic heat provision' can either by of subtype 'gas furnace' or 'biomass furnace'), and both are sub-classes of the *Concept* class. This *Concept* superclass is comprised by compositions to other concept classes: Attribute of a concept, and *Relation* between the concept and another one, both of which are also part of the ontology. A third composition links the *Concept* class to the *Instance* class, with which specific manifestations of the concept class can be specified as facts (e.g. a concrete individual of type human actor, or a concrete appliance of type gas furnace). There can also be instances for relations and for attributes,

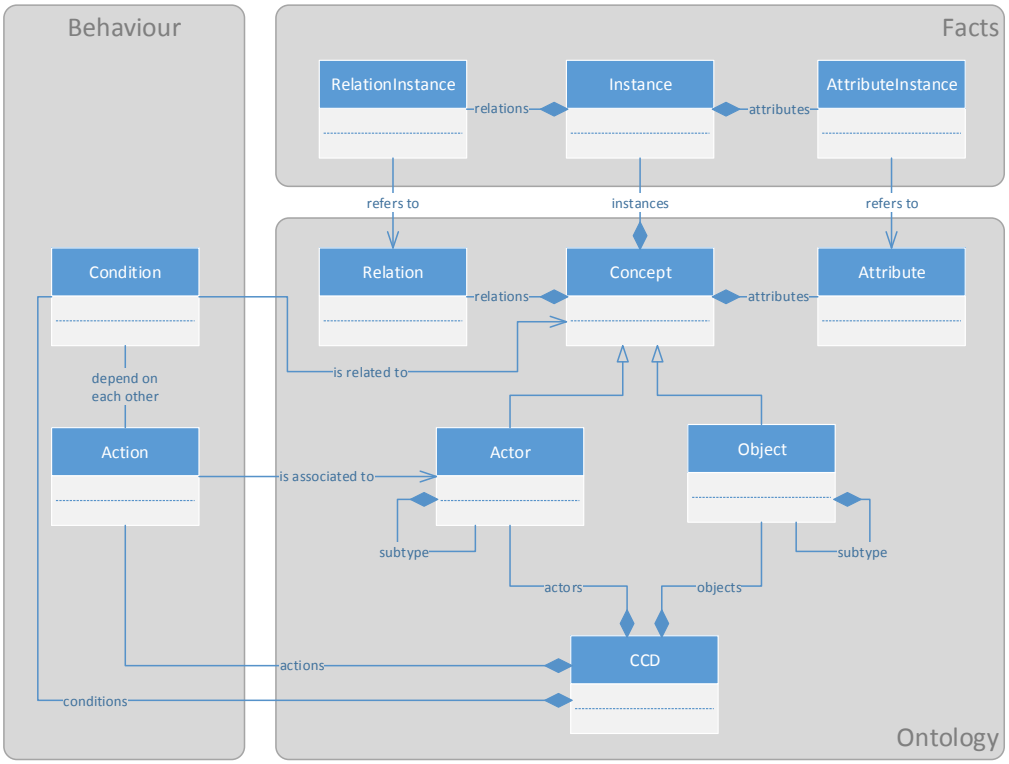

Figure 3.4: Core elements of the CCD meta-model (simplified version of meta-model published in (Scherer et al., 2013b))

defining the concept instance further. As mentioned before, the CCD meta-model foresees two concept classes for specifying behaviour: *Condition* and *Action*. Each action is associated to an actor type, and can only be applied if some conditions hold. A condition is related to *Concept* in a way that either some facts related to a concept must exists in order to make an action possible (pre-condition), or — as a result of an action — existing facts are modified or new facts created (post-condition).

To make this more tangible, in the following subsection examples are shown how to apply the CCD language as specified by the meta-model.

Tool support

Effective conceptual modelling requires a carefully designed toolbox integration. It is important to particularly accommodate the requirements of modelling experts, since they can be considered the main user group of this part of the toolbox. In OCOPMO, the Eclipse Integrated Development Environment (IDE)[32] was selected. Eclipse as a popular platform for Java developers

[32] `http://www.eclipse.org/`

also in the social simulation domain (e.g. a popular simulation tool — Repast Simphony (North et al., 2013) — is based on Eclipse) was easily adopted by modellers involved in OCOPOMO.

From this user perspective, a variety of functionality should be made easily accessible:

- graphical creation and editing functionality for user-friendly CCD development following the language description specified by the meta-model;
- download of scenarios and background documents from the collaboration platform in order to create a local evidence base (working copy);
- annotation functionality for linking phrases from texts in the evidence base to the different elements of the CCD;
- access to code transformation functionality, as described in subsection 3.3.3.

A software solution that provides all this functionality was developed under the name CCD Tool (Scherer et al., 2013b). It is composed of a number of so called Eclipse features, each consisting of a number of plug-ins, providing a specific subset of functionalities in the shape of smoothly integrated parts within the Eclipse IDE. Smooth integration in this case is achieved by building the features on base of Eclipse standard components, e.g. for editor views, creation wizards and menu items.

The core part of the CCD Tool is the graphical editor, closely linked to the annotation editor. The most important user interface components of the editor are:

- four actual editor features: one general CCD editor and three specialised views on the CCD: an actor-network diagram, an action diagram and a (less important and rarely used) object diagram;
- an outline view;
- customised property views for each CCD element.

The screenshot in Figure 3.5 shows the general CCD editor in the upper part, where a scenario text is loaded and several text phrases are highlighted in different colours. The colour refers to the type of CCD element that is annotated with the phrase, e.g. red highlights refer to actors, purple to relations. The lower part of the window shows the CCD outline view, where a few actor types can be seen. For the actor `RegulatoryOffice`, the sub-elements are visible, which are a text annotation for the actor type, an (the only) instance of the actor type with the name 'URSO' (i.e. the concrete regulatory office from Slovakia in this example), and a relation between the actor `RegulatoryOffice` and the object `PriceChange`. For this relation, another text

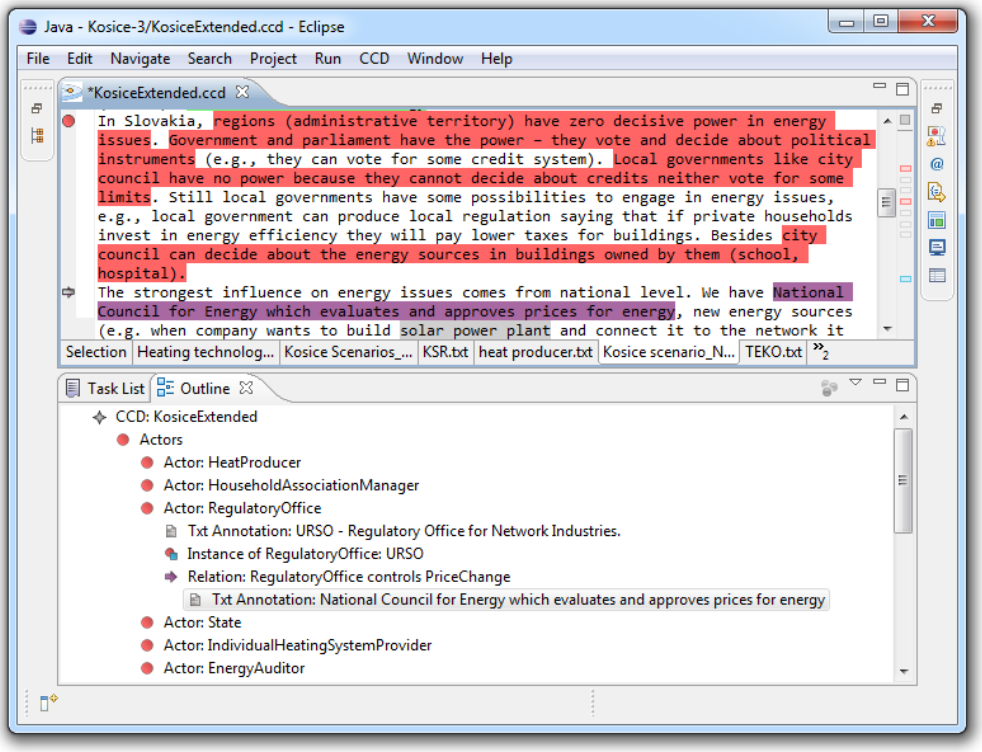

Figure 3.5: The same stakeholder scenario as shown in Figure 3.3, displayed in the Eclipse-based CCD Tool with annotations (upper part), together with a tree visualisation of parts of the CCD (lower part)

annotation is visible, referring to the purple phrase in the document above. As the screenshot reveals, the CCD outline (which is at the same time the general graphical representation of the CCD) is a tree view, comprising all types of language elements defined in the meta-model.

One of the specialised views on the CCD — a small and simple fragment of the ontology in form of an actor-network diagram for this CCD — is shown in Figure 3.6. There, again, the actor `RegulatoryOffice` actor (denoted with the symbol ●) is displayed with an assertion about its responsibility to control price changes (in this example for heating energy). The `PriceChange` is modelled as an object (symbol ■) with three attributes (symbol ✚): the direction of the price change (reduction or increase), a proportional and an absolute value of the price change. The control function of the regulatory office is modelled with the relation `controls`, as also contained in the tree view in Figure 3.5 as child entity of the `RegulatoryOffice` actor. In the diagram, there is also another actor `HeatProducer`, which also has influence on price changes, as he `makes` the price change, i.e. defines the actual price. This actor

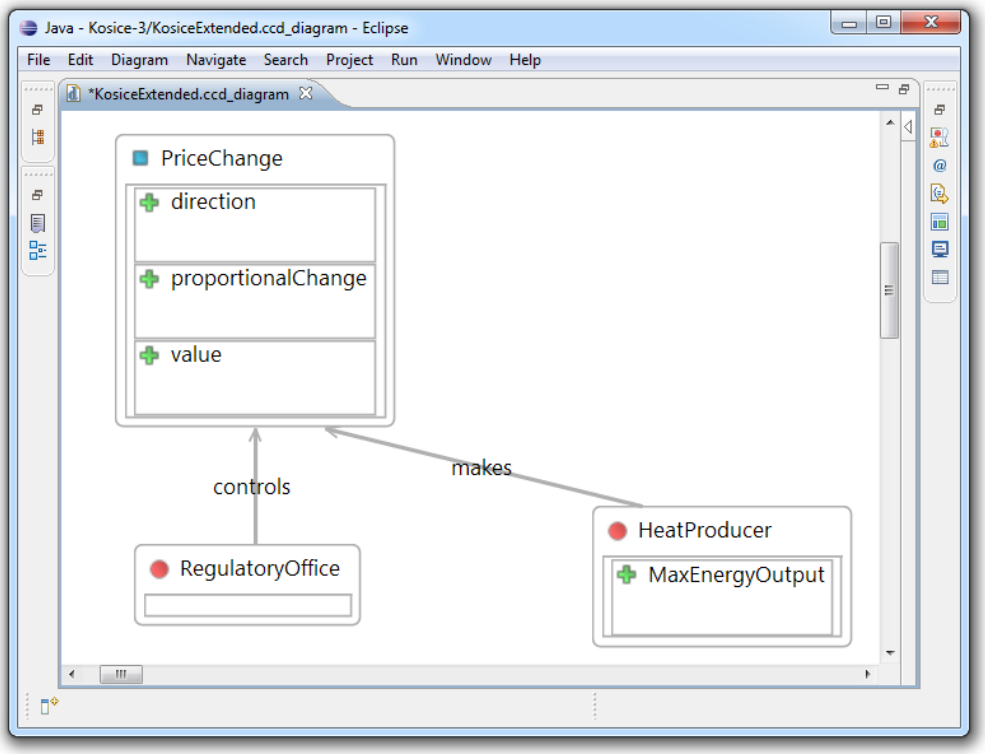

Figure 3.6: Actor network diagram example based on the CCD ontology (small fragment from a larger model)

has an attribute indicating its maximal energy output.

An equally minimalist example for a behaviour definition is given in Figure 3.7: a small part of the action diagram for this CCD. Here the single action `heat producer changes prices` is shown (with symbol). This action is assigned to the `HeatProducer` actor and has two preconditions (symbol): Firstly, as a reason for increasing the energy price, it can be assumed that the `price of energy source rises`. Secondly, the regulatory office `URSO agrees to price change`. As a result of the action, the `heat price changed` postcondition (also with symbol) is then valid. In the lower part of the condition boxes a formalisation is displayed which can be specified to express the relation of a condition to a concept (according to meta-model), relying on the concept of variables. For example, the expression '`?change deliveryPrice >0`' expresses that the variable `?change` related to `deliveryPrice` must have a positive value to fulfil the precondition `price of energy source rises`.

From a more technical perspective, in addition to the user interface functionality the CCD Tool constitutes the hub for traceability between the evidence base on the 'input side', and the formal model on the 'output side'. For this

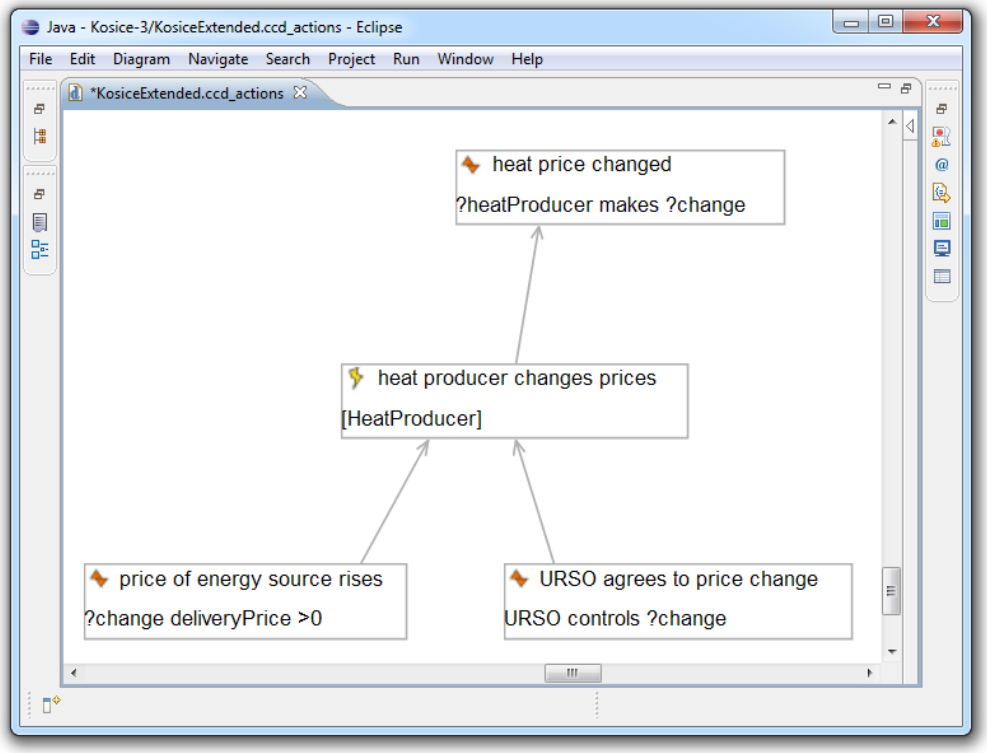

Figure 3.7: Action diagram of a small part of the CCD behaviour description

purpose, each element of a CCD has additional information attached: On the one hand annotation phrases with links to the original documents (incorporating an URI[33] and a locator of the phrase within the document), on the other hand a unique identifier (UUID[34]) with which the CCD element can be unambiguously referenced. With these two elements, the subsequent components in the tool chain can gain access to the trace information by referring to the UUID of related CCD elements. This is the foundation for traceability in the approach proposed by OCOPOMO, as discussed in more detail in section 3.4 (for the user perspective) and in Chapter 10 for technical details.

The implementation of the CCD Editor is based on the Eclipse Modelling Framework (EMF)[35], thus it is furnished with inherent support for code generation. The meta-model shown in the previous subsection is just the simplified Ecore model for this component. Consequently, the graphical description language is implemented on base of the Eclipse Graphical Modelling Framework, GMF[36]. The information of the general CCD as well as for the different views

[33]Uniform Resource Identifier, (Berners-Lee et al., 2005)
[34]Universally Unique Identifier, (Leach et al., 2005)
[35]`http://www.eclipse.org/modeling/emf/`
[36]`http://www.eclipse.org/modeling/gmp/`

can be serialised and stored in files relying on a specific XML format and, thus, can easily be processed by other external tools. An example for such a tool is the CCD Viewer mentioned in Table 3.1.

The interface features to the collaboration platform and to code transformation can be seen as supplementary components. The former will not be further detailed here, but is described in (Bednár and Hartenfels, 2013). The latter is subject of the following subsection.

3.3.3 Model Transformation

This section describes the bridge between the phases 'Conceptual model development' and 'Policy model implementation'.

Conceptual modelling on its own can have beneficial effects on the quality of the implementation of the modelled system, related to a better understanding of the relevant requirements — as discussed in the previous section. The true benefits, however, will be acted out if the formal aspects usually contained in the conceptual model are automatically transformed into code that makes up at least a basis of the actual policy model implementation. Here comes — as mentioned in section 2.2.3 — the trade-off between the degree of formalisation within the conceptual model and the portion of the implementation that could be generated into play: The more formal aspects are already included in the concept, the more code can be generated. According to the model-driven architecture approach, the entire implementation could be conceptually specified. Although this might work well for some standardised software solutions[37], this is difficult to achieve for complex simulation models for the following reasons:

Firstly, the requirements for the software representing a simulation model are only in very rare cases clear enough at the specification phase in order to achieve a comprehensive specification. As mentioned before, this is related to the fact that the system to be implemented is far too complex to be represented anywhere nearly complete, so the challenge for the modeller is to find a best possible compromise, a suitable degree of abstraction. This process of abstracting is typically an iterative approach: Starting with a strongly simplified abstraction, in the course of time more and more relevant elements are added in several iterations of implementing and test-running the system. The main problem with conceptual modelling here is, that it is often unknown prior to tests with the (first) implementations, by which means certain facts need to be realised in order to reflect the real system in an appropriate way.

Secondly, unlike in typical industrial applications where for repetitive problems best-practices of software engineering or even standardised and readily implemented software frameworks are provided, such building blocks for simulation models are available only for solutions of either very specific or rather trivial problems. This holds primarily due to the fact that the scope of the

[37] See for example the MDA criticism by Picek and Strahonja (2007) and more recent developments in Model-Driven Development in (Pastor, 2017).

problems to be solved by software is closely coupled to the particular subject matter of the system to be modelled. For example in the case of social simulation, each social system to be simulated requires different aspects of behaviour of the individuals as part of this system.

However, model implementation can to some considerable extent profit from code generation. This is because the scaffolding process of the model implementation can be done almost completely automated, resulting in a proper program structure (a 'model skeleton') based on 'best practice' architectures — which, however, are available for agent-based simulation.

Code generation from the CCD

Code generation as result of the model transformation will be demonstrated with the example of a type of simulation model introduced in the following section 3.3.4: Structural elements (like agent classes) are represented in an object-oriented way (i.e. Java classes), while the working memory (i.e. facts) and behaviour are represented within a declarative rule engine. The target language of the transformation is outlined as a high-level meta-model of the architecture in Figure 3.10), while the source language is defined by the CCD meta-model of Figure 3.4. The transformation rules shown in Figure 3.8 refer to elements defined in these two meta-models.

The *CCD* itself logically constitutes the bases for the simulation model, so in the transformation it becomes the main *Model Class* of the imperative Java part. For each *Actor*, a specific Java class (the *Agent Class*) is generated, each representing an agent. Code for instantiating these agent classes (as *Agent Instance*s) is added to the model class, originating from the *Actor Instance*s in the CCD.

The actors also find their representations in the declarative code. For each actor, a file with agent-specific declarative definitions is generated, containing information (i.e. attributes) about the actor itself (in form of DRAMS attribute facts), about *Relation*s between the actor and other concepts (in form of DRAMS relation facts), as well as privately known *Object*s of the actor (in form of DRAMS object facts). For each of these *Fact*s, the associated data type specifications (in form of DRAMS *Fact Template*s) is generated as well. Global knowledge, i.e. concepts and relations known to all actors is reflected in global DRAMS definitions, which are of the same kind as for agents.

In contrast, behaviour is always associated with actors, so that the actions are always transformed into agent-specific DRAMS *Rule*s. For the conditions, hints for possible declarative language constructs are given in form of comments.

An important feature not shown in the diagram is that for each transformed CCD element, annotations and comments are attached to the generated target elements. These annotations contain the CCD UUIDs for automated traceability processing, but also the content of CCD element comments and text

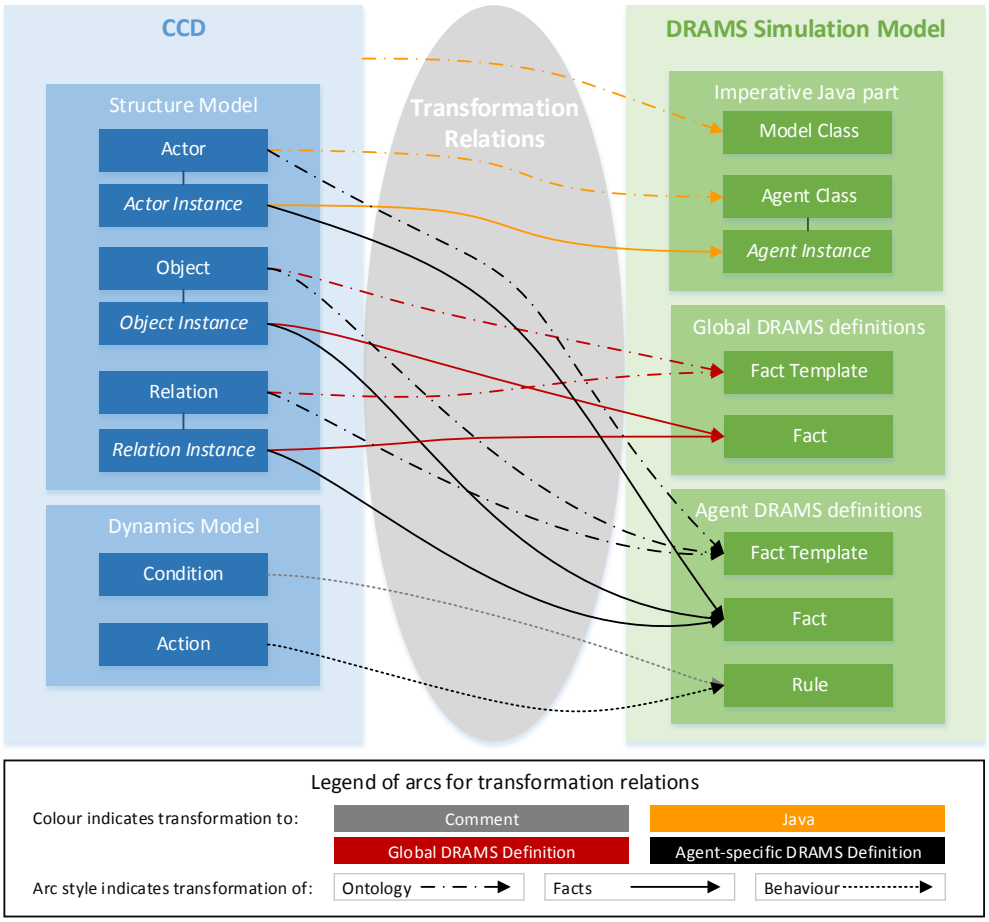

Figure 3.8: CCD2DRAMS transformation relations

annotations. The latter constitute a major part of the simulation model documentation.

Tool Support

The CCD2DRAMS tool for realising the transformation described above was implemented for the OCOPOMO toolbox as a prototypical direct model-to-text transformation (Lilge, 2012). Both the meta-models of the CCD and of the simulation environment were taken into account, but since no meta-model for concrete simulation model implementations can be provided (besides the architecture meta-model, as reasoned above), only the meta-model of the CCD is directly reflected in the transformation rules. As a consequence, there is no bi-directional transformation possible: While changes in the CCD can be transformed into changes of the simulation model code (without disturbing manually edited code portions), changes in the code are not directly adopted

in the CCD.

CCD2DRAMS is based on the Acceleo code generator[38], and is tightly integrated with the Eclipse modelling environment. It consists of a set of scripts, implementing (a superset of) the transformation rules shown in Figure 3.8. The additional transformation rules basically cope with appropriate inclusion of CCD conditions (using a concept of variables), comments and annotations for traceability. The result of the transformation is a complete Eclipse project, containing all the necessary Java classes and DRAMS definition files — each containing the appropriate stubs for methods and other language elements — in a way that the model can immediately be executed (without any sensible outcomes of course, as the stubs need to be manually filled with code; see next section 3.3.4).

As an example for generated code, Figure 3.9 shows a screenshot of the Eclipse IDE, with a DRAMS editor in the upper part of the window, displaying a rule stub (identifier `defrule`) named `heat producer changes price`, associated with the agent `HeatProducer`. The three comment lines in the left-hand side of the rule (above the `=>` symbol) are related to the facts constituting the pre-condition of the rule, while the `TODO` comments are just placeholders for the real code to be developed manually. Another important part is, as mentioned before, the comment section above the rule, consisting of the related action name, the link UUID of the CCD action and two text annotations, i.e. phrases from documents of the evidence base. For comparison, the CCD action from which this rule stub was generated can be seen in the CCD Editor shown in the lower part of the window.

3.3.4 Simulation modelling

With the previous steps of conceptual modelling and model transformation performed, the 'Policy model implementation' phase can be entered. The starting point for implementing the simulation model resembles MDA realisations, as — for instance — typical 'scaffolding' activities (like creation of Java classes for the model and the different agent types) are performed automatically. However, the amount of work taken over by the model transformation varies with the degree of detail with which the conceptual model has been equipped, as discussed earlier. Aim of this process step is to ultimately create the simulation model as a formal endpoint of the abstraction process. A simulation model is a runnable computer program which takes configuration parameters as input data, and produces different kinds of results. These results need to be in line with the specification in the conceptual model, and are influenced by the input parameters. Dependent on the simulation modelling paradigm, the simulation runs are either deterministic (i.e. each run of the simulation model with identical input parameters produces identical results), or influenced by stochastic processes (i.e. for identical input parameters different results within

[38] `http://www.eclipse.org/acceleo`

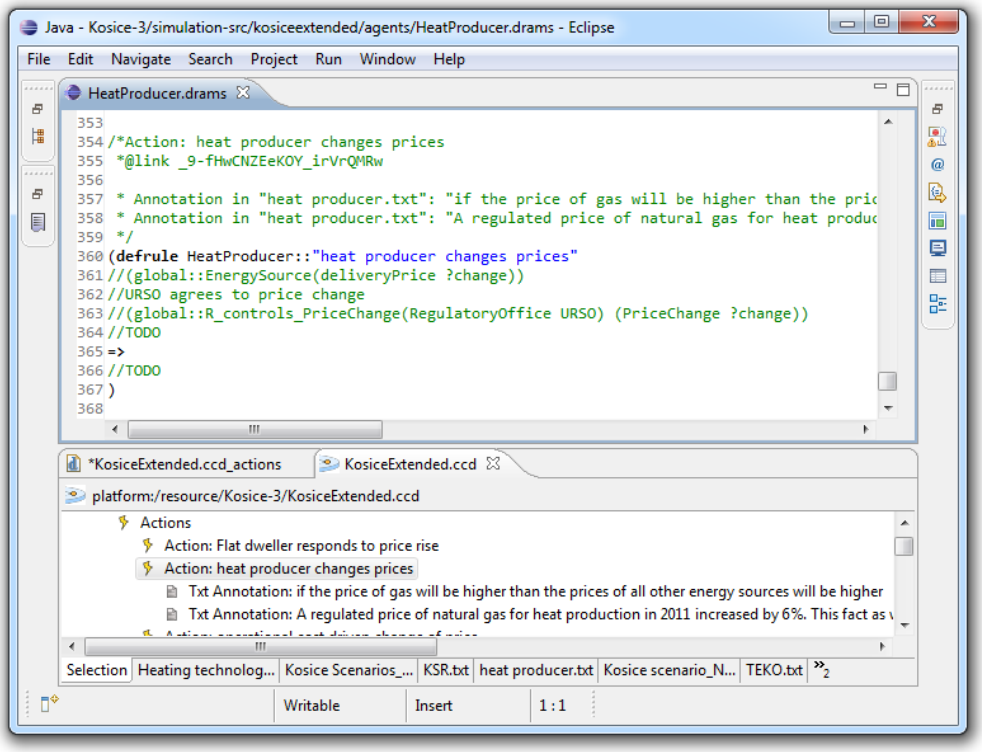

Figure 3.9: Example of a DRAMS rule stub, generated by CCD2DRAMS (upper part) and the related action as part of the CCD tree view (lower part)

a certain space are possible).

In this abstraction process right up to the simulation model code — although based on documents in the evidence base — information gaps might appear which somehow need to be filled in by the programmer, in order to let the simulation model produce sensible behaviour. For example (and as mentioned in Chapter 1), simulating behaviour of human beings includes aspects that 'exceed' the often merely rational deliberations considered in the previous data collection and conceptual modelling phases. Model implementers basically have two options to fill such information gaps:

- They might go back to the collaborative data collection phase, trigger a new round of scenario generation and discussion with focus on the missing information. This feedback has then to be integrated in the CCD, and (after a new transformation) becomes available as new facts or rules in the simulation model. This is certainly the preferable approach in case of crucial aspects that were overlooked previously.

- Much more frequently the information gap is related to an aspect for which the involved stakeholders cannot contribute with their expertise.

In such cases, the implementers can either trust their own experience and introduce 'cognitive heuristics'[39] to the model, or they must obtain the missing information by desk research or from external expert knowledge. Here, also a decision has to be taken how to include these aspects in the model: either by directly implementing the aspects in the simulation model, or by introducing new concepts into the CCD.

These considerations will become relevant in the following chapters 4 (when adopting the modelling process for GLODERS) and 5 (when developing the use case model). The remainder of this section will instead go a step back and introduce the means by which the simulation models are implemented in the OCOPOMO approach, and therewith complement the state of the art on simulation given in section 2.3.

Imperative models with declarative rule specification

In the context of OCOPOMO, the term simulation model refers to a formalised policy model, implemented with an agent-based approach in a declarative, rule-based fashion. As declarative rule engines specially tailored to the specific requirements were not available at the time when the project started, it was decided to embark on a hybrid design that promised to take advantages from a dedicated simulation tool based on the imperative programming paradigm, in combination with a declarative rule engine (Lotzmann and Meyer, 2011b).

This proposed architecture is reflected in a high-level architecture meta-model of a simulation model implemented in this way, as shown in Figure 3.10. According to this meta-model, a simulation model is composed of an imperative part, reflecting the structure of the model, and a declarative part, representing the behaviour and 'knowledge repository'. The imperative part consists of a (single) model class and an arbitrary number of agent classes, which are managed by the model. The declarative part, though, provides world knowledge to the model, as well as individual (private) knowledge to the agents, and determines their behaviour. The knowledge representation within the declarative part includes fact templates — defining the structure and data types allowed for facts — and the actual facts. The behaviour is formulated by rules which, in general terms, process the facts in some way.

The purpose of simulation modelling within OCOPOMO is stated in (Scherer et al., 2013a) as to be "designed and implemented in order to explore the individual actions and combinations of actions that are believed to be available to governments for the purpose of achieving specific and well-formulated objectives. It enforces the specificity and clear formulation of available actions and also clear, precise and well formulated statements of the conditions in which the actions might be taken and the consequences of those actions in the specified conditions. The model produces output in the form of [...] statements

[39] Cognitive heuristics are realisations based on reasonable assumptions, as e.g. illustrated in (McCollough et al., 2014). These are sometimes also casually called 'magic facts'.

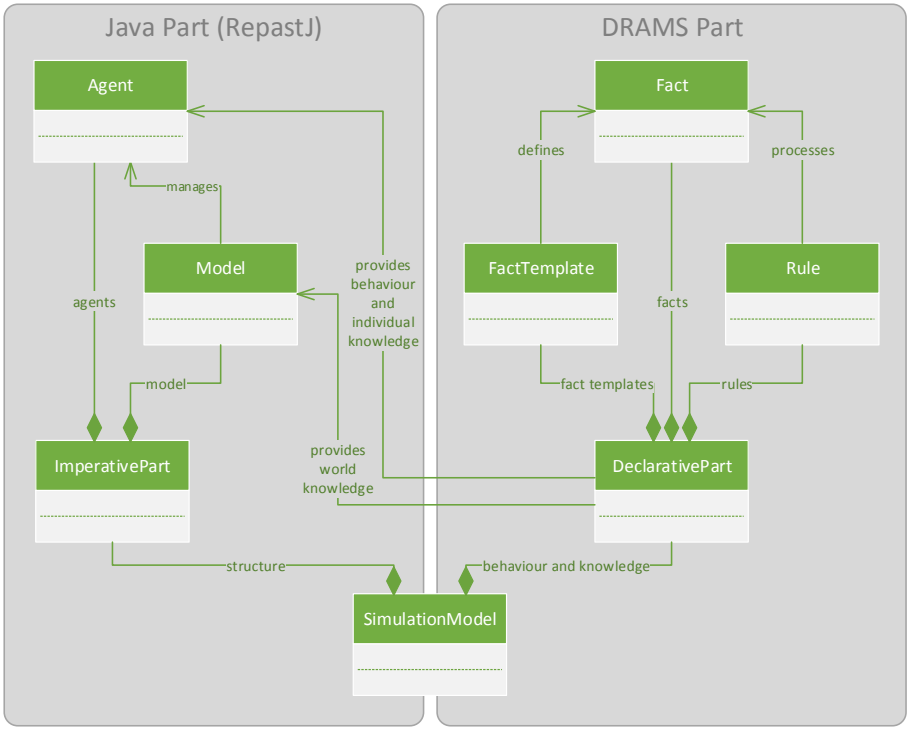

Figure 3.10: Simplified meta-model of a typical DRAMS simulation model as supported by the CCD2DRAMS tool

that describe sequences of events including decisions and the outcomes that emerge during simulation experiments with the model."

Table 3.2: Procedure of implementing a DRAMS simulation model (as suggested in (Scherer et al., 2013a))

Step	**Model skeleton generated with CCD2DRAMS**	**Model set up from scratch**
1.	The generated Repast model and DRAMS files should be inspected for consistency and completeness.	A Repast model class must be defined, which creates all agent instances (for which the desired agent types have to be defined, see step 2), initialises the global fact base and handles the Repast time steps by triggering the rule scheduler. An abstract model super class providing the DRAMS related code is available in the DRAMS distribution, thus only model related aspects have to be added.
2.	Optionally, Java classes for model or agents can be adapted or extended.	Classes for all designated agent types must be created and all necessary functionality should be implemented. Similar to the model super class, an appropriate agent super class is delivered with DRAMS.
3.	The rule definition stubs need to be elaborated by specifying the Left Hand Side (LHS) clauses (for the condition part) and the Right Hand Side (RHS) clauses (describing the action part). Usually, a number of additional declarative code definitions (e.g. type definitions for intermediate results, supporting rules) need to be added.	For each agent type, code for the declarative model part has to be written in the related DRAMS files. Firstly, the fact templates have to be specified, after that the initial facts can be asserted to the fact base, and finally the rules can be written.
4.	The declarative model part can be checked for consistency, using a visualisation of the automatically generated dependency graphs. Furthermore, testing and debugging procedures should be performed by executing the model using the Repast user interface, possibly with a reduced number of agent instances.	
5.	If the model is running as expected, then additional code for creating and storing simulation outcomes can be implemented.	

Continued on next page

Table 3.2 – continued from previous page

Step	Model skeleton generated with CCD2DRAMS	Model set up from scratch
6.	Productive runs of the full-scale simulation model can be carried out, again using the Repast interface. Textual logs and numerical data are generated and stored according to the definitions made. XML based result files can then be analysed with the Simulation Analysis Tool.	Productive runs of the full-scale simulation model can be carried out, again using the repast interface. Textual logs and numerical data are generated and stored according to the definitions made, but no traceability information will be available.

The process of implementing a simulation model using RepastJ as simulation platform, extended by DRAMS as the declarative rule engine, can be structured into a procedure comprised of six steps. This procedure is shown in Table 3.2 for both cases where the model skeleton is created by CCD2DRAMS, and for the manual set-up of the model from scratch (to demonstrate once more the contribution of code generation; see also Lotzmann and Meyer (2011b) and Lotzmann and Meyer (2011a)). The proposed steps anticipate the usage of certain tools and methods, which will be explained later in this chapter.

Tool support

The core of the simulation environment within the OCOPOMO toolbox is the RepastJ version 3.1 (North et al., 2006) simulation tool covering the imperative part, with DRAMS as extending framework for the declarative part. Both are Java-based tools, so interoperability is straight-forward, and as a consequence the language of the imperative simulation model part is also designated to be Java. In contrast to RepastJ as an existing well-established simulation tool, DRAMS is a new software developed within the OCOPOMO project and at the same time constitutes the major contribution of this PhD thesis. Hence, several chapters are dedicated to this tool and will present DRAMS both from the application (Chapter 6) and the technical perspective (Chapters 8 and 9).

From the modeller perspective, DRAMS as both a development tool and as part of the simulation execution platform needs to maintain pillars from both sides. I.e., on the one hand a user interface integrated in the Eclipse IDE is necessary to create and edit the declarative definitions, on the other hand a user interface needs to be provided to control and supervise simulation runs, coupled with the RepastJ user interface. For both pillars respective components are provided: Figure 3.9 shows a screenshot of the DRAMS editor Eclipse feature, while the screenshot in Figure 3.11 displays a so-called Data

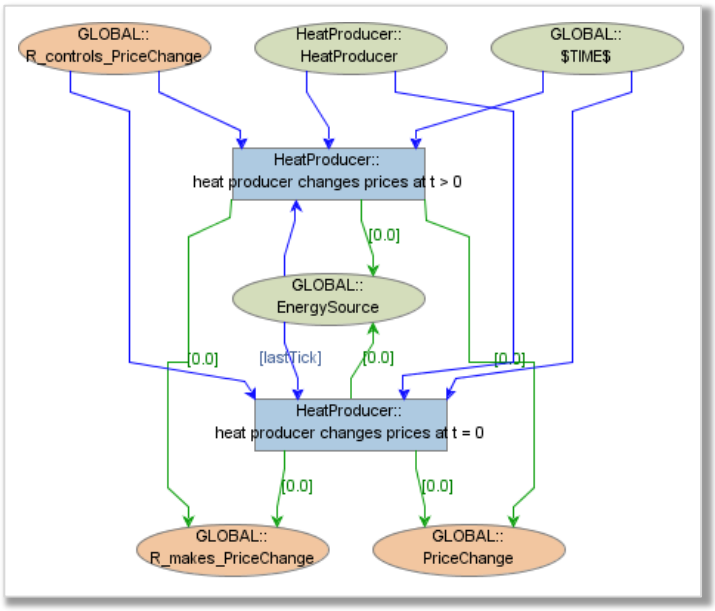

Figure 3.11: The Data Dependency Graph with the rules that have been implemented from the generated rule stub as shown in Figure 3.9

Dependency Graph (DDG) generated by DRAMS as a visualisation of the actually implemented model in running state.

The DDG shown in this figure reflects the implementation of the previously shown actions (and transformed rule stub) `heat producer changes price` presented in 3.3.3. What can be read from the diagram is that the original rule stub has been split into two actual rules (the rectangular shaped nodes), while the oval nodes stand for facts representing the pre- and post-conditions for the rules. Section 7.3.4 provides more details on this diagram type and related topics.

These tools are also helpful for model verification, i.e. ensuring best possible correctness. In the first place, this involves a consistent simulation model behaviour, so that the model behaves in plausible ways for sensible ranges of input parameter settings. Such a verified model can then be handed over to the next process phase, where productive runs of the model produce analysable and validatable results.

3.3.5 Simulation result generation and analysis

This section comprises the phases 'Simulation and generation of model-based scenarios' and 'Evaluation and validation of evidence-based vs. model-based scenarios'.

Having a simulation model implemented and verified, a phase of well-planned experimentation and result analysis can be entered. The aim of this phase is

validation, which takes place in two different steps. The first step is the internal validation, i.e. it must be ensured that the simulation results do not stand in contradiction with information (e.g. from the evidence base) incorporated in the model. This involves experimenting with the model, performing multiple simulation runs, collecting the outcomes produced by the simulation, checking them for plausibility and trying to identify patterns in simulation results. The second step is then to condense the experiences collected during experimentation, in the case of the OCOPOMO approach in so-called model-based scenarios, narratives exemplarily describing typical simulation results, related to the identified pattern. The actual (external) validation is then again a collaborative endeavour, as these results are discussed with the stakeholders, in order to be approved (or denied) by them. These two steps — subsumed under the term 'productive simulation runs' — are elaborated in the following subsection.

Between these two validation steps an important point of decision in the modelling process is reached. If the internal validation fails, usually the corrective measures are taken by stepwise going back to either the simulation modelling or the conceptual modelling phase, according to severity of the deviation. An external validation which was not approved in a discussion with stakeholders and domain-experts, on the other hand, triggers a new iteration of the modelling initiative with a collaboration phase for creating new or adapting the existing narrative scenarios.

Productive simulation runs

Productive simulation runs require executing the simulation model with different input parameter settings. Before — as part of model verification — the parameters for which the simulation results change in a significant and replicable manner have to be identified. This can be achieved with a sensitivity analysis (Chattoe et al., 2000) based on quantitative methods, or with a qualitative approach. The latter involves analysing and interpreting the simulation outcomes by a human 'simulation analyst'. This approach has been chosen for all the models developed referred to in this PhD thesis. A detailed example of this step is given in Chapter 7.

The outcome of the qualitative result analysis needs to be condensed and made digestible for stakeholders, i.e. it must be transformed into the language of the application domain (or even in everyday speech). The resulting artefacts are referred to as model-based scenarios, as the style of these documents should be similar to the scenarios generated by the stakeholders. Model-based scenarios can be enriched with diagrams and (hyper-) links to some kind of evidences, creating the bridge to the raw simulation results and — by applying traceability — to the documents in the evidence base. A comprehensive description of this type of simulation analysis is given in Scherer et al. (2013a).

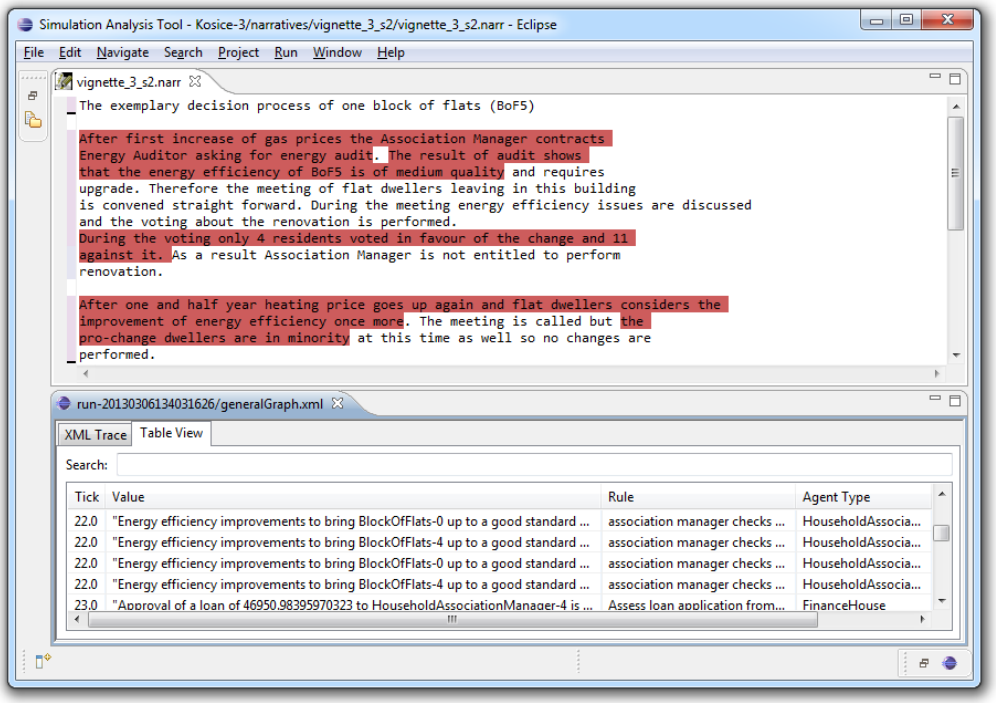

Figure 3.12: Eclipse-based Simulation Analysis Tool, which allows editing of model-based scenarios (upper part) face-to-face with the simulation outputs (lower part)

Tool support

The creation of model-based scenarios is supported by another Eclipse feature, the Simulation Analysis Tool (Furdik et al., 2013). Figure 3.12 shows a screenshot of a typical application situation for this tool: In the upper part of the screen a text editor for writing the scenario is situated, below in the lower part a table is provided by the tool that displays the simulation log in XML format as generated by DRAMS in a human-readable manner. With a simple user action, a phrase of the scenario can be linked to an arbitrary entry of the simulation log, hence creating a trace to the evidence base.

The completed model-based scenarios are then published in the collaboration space. Hence, another connection point to the Alfresco collaboration platform has to be established. For transferring the data from Eclipse to Alfresco, the Simulation Analysis Tool is bundled with a respective upload functionality — the Data Repository Client in Figure 3.2.

3.4 Tracing back simulation results

As an overarching layer above the OCOPOMO process and the related tools, the technical means for preserving traceability across the artefacts and transformation steps is one of the central features of the OCOPOMO toolbox. This sections is aiming to briefly illustrate the underlying mechanisms, to be concretised and substantiated later in this PhD thesis.

The rationale for introducing traceability in the OCOPOMO process is pointed out by Lotzmann and Wimmer (2013a), following the discussion in section 2.3 (in subsection Evidence-based simulation models). The purpose of traceability in this context can be summarised as follows:

> "Traceability is a key element in ensuring openness and transparency in the OCOPOMO policy development process. Therewith, good governance principles are implemented in policy modelling. Traces and provenance are also important auxiliary means for policy modellers, by helping the experts to better understand complex interrelations of policy aspects and how informal data elements feed into a formal model (provenance). Hence, traces are a basic instrument for model exploration and easier understanding of the structure of a simulation model, for simplification and visualization." (Lotzmann and Wimmer, 2013a)

The technical realisation of traceability is the subject of Chapter 10. Focus of this section is to give an overview how the different artefacts are linked together in order to maintain a chain of links, from the model-based scenarios back to the evidence base consisting of narrative scenarios by stakeholders and background documents. Figure 3.13 shows again all the artefacts created in the different process steps as introduced in Figure 3.1, with the link as pointed out in the previous sections.

The consistent policy model as final result of the modelling process can be seen as the container for all other artefacts: the initial and stakeholder scenarios with related background documents, the CCD as conceptual model, the formal policy model in form of an executable simulation software, and the simulation results formulated as model-based scenarios. The latter three types of artefacts, i.e. the models and the results, contain meta-data to identify the sources from which they are derived. For each element of the CCD, this is an attached text annotation referring to the source file document and the position of the relevant phrase inside this text. The CCD furthermore creates a unique identifier (UUID) for each element, which then can be used by subsequent modelling artefacts to hook up into the trace chain. In case of DRAMS simulation models this type of link is established by tags annotated to each declarative language construct, that contain UUIDs to the related CCD elements. This information is also attached to the simulation outcomes and, hence, can be incorporated in the model-based scenarios via hyperlinks. In the

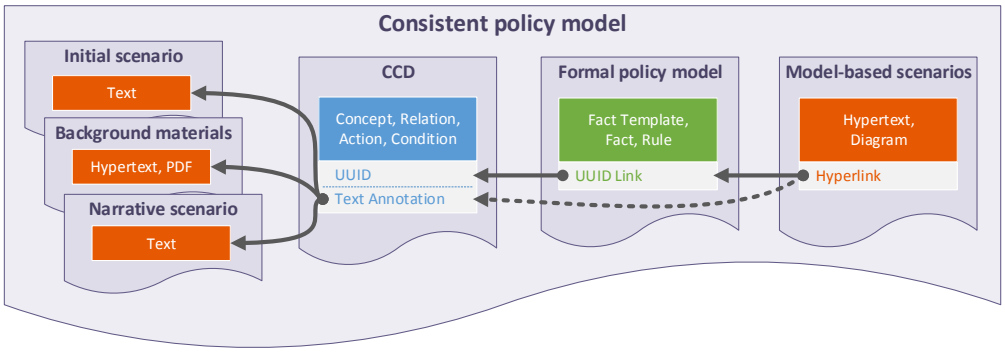

Figure 3.13: Artefacts of the OCOPOMO process with traceable links

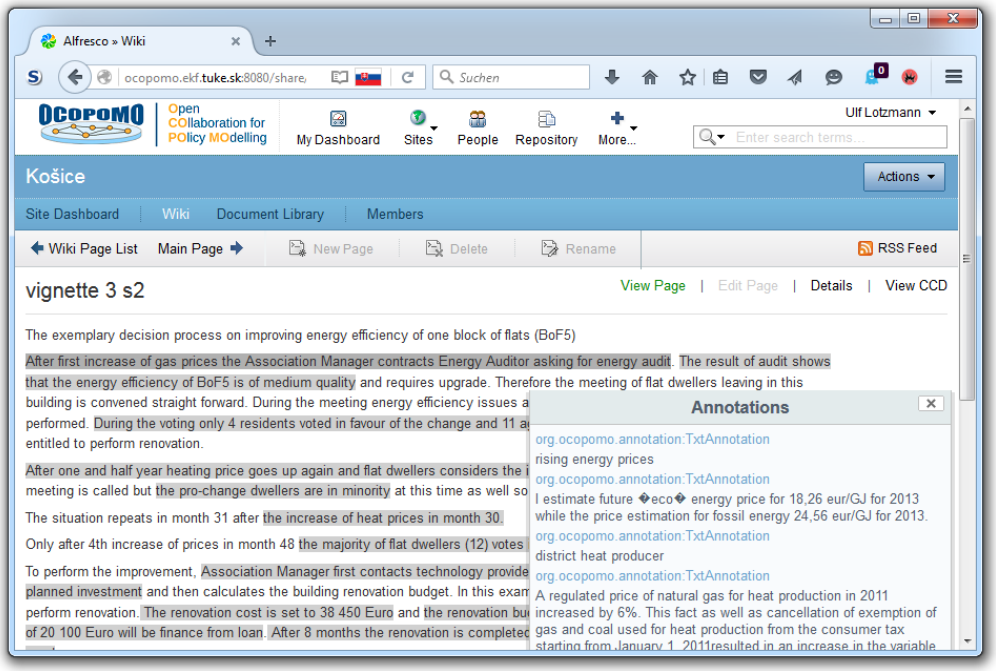

Figure 3.14: Example for a model-based scenario, presented in the Alfresco collaboration space, with box showing the annotations to the evidence base for a selected phrase

presentation of these scenarios the UUIDs are replaced with the text phrase behind the UUIDs, taken from the CCD.

Figure 3.14 gives an example how the presentation of a model-based scenario looks like in the Alfresco collaboration space. These scenarios are created

as hypertext documents and might include diagrams (not shown in the screenshot) and hyperlinks. The hyperlinks in the example are the grey highlighted text phrases. A click on such a link opens an 'Annotations' windows with information on the relevant traces: the annotation phrase linked to the CCD element and the link to the document in the evidence base.

3.5 Conclusions from the OCOPOMO project

The OCOPOMO project introduced a new approach to policy modelling and simulation — as a sub-discipline of social simulation — by applying methods known from software engineering, specially adapted for this domain. The resulting OCOPOMO process with its core features

- qualitative method for data collection (scenario method in this case),
- conceptional modelling with model-driven application development (i.e. including code generation) and
- traceability

provide a solid basis for conducting complex evidence-driven modelling tasks. This process is also flexible enough to be modified in various phases, without invalidating the beneficial properties proven in the OCOPOMO pilots. Such a modification will be looked upon in the subsequent chapter, for which then an elaborate use case is studied in Part II of this PhD thesis. From the perspective of this thesis, the OCOPOMO project does not only provide the methodological framework, but also a part of the developed toolbox, in particular (and in order to support traceability from a technical standpoint) the CCD Tool and DRAMS together with CCD2DRAMS.

Aside from the modifications and examples given in the next chapter, the restrictions and possible extension points of the OCOPOMO outcomes are discussed in the following.

The modelling and simulation software tools developed in OCOPOMO are exemplary prototypical implementations, although certainly not just proof-of-concept prototypes. These tools have been and are used for several other projects and purposes. One of the compromises made in OCOPOMO — and hence posing a restriction — is the model-to-text code transformation implemented in the CCD2DRAMS tool. This approach allows the adaptation of formal program code on conceptual model changes to some degree, but prohibits the opposite direction of trailing code changes in the conceptual model. A proper model-to-model transformation approach would help to overcome this restriction.

The main restriction, however, regards the consistent conceptual description (CCD), in particular the degree of formalism that can be attached to conceptual models. The current CCD does not provide enough information

for generating comprehensive simulation model code, e.g. some parameters of facts cannot be expressed (target agent fact base, fact permanence, comprehensive rule logic etc.).

Possible solutions for these restrictions would require to equip the CCD with simulation model relevant information. This imposes the problem that the CCD would become dependent on the target language, while the concept of CCD foresees target code independent models which should be transferable into arbitrary programming languages. Therefore it would be needed to break the transformation process into two distinct parts by inserting a transformation model, a language-independent formalisation model.

Another promising and important objective of future research is the opening of the OCOPOMO development process and the associated tools for additional methods of data collection and analysis and simulation paradigms, beyond the first steps (with respect to data analysis) described in Chapter 4. An important contribution by Majstorovic et al. (2015) provides a "comparative analysis of simulation models utilised in the field of policy-making" with the goal to enable "hybrid simulation models" combining "different modelling theories".

Chapter 4

Linking the modelling process with data analysis

4.1 Introduction

This chapter presents a customisation of the modelling process introduced in the previous Chapter 3 with the aim to make it applicable for a specific modelling task, where the discussion on criminology in section 2.4 comes into play. Thereby, the different process phases are underpinned with detailed sub-processes, and in particular the domain-knowledge collection and the simulation result generation and analysis phases are tailored for incorporating data analysis and qualitative research methods. This specialised modelling process constitutes the methodological approach applied in Part II.

The customisations and major parts of the content presented in Part II are results from another research project (GLODERS). The author of this thesis participated in this research project primarily as simulation modelling expert, together with a data analysis expert (Martin Neumann) and other teams and stakeholders. Substantial parts of his contributions are presented in this chapter (regarding the model implementation phase), and constitute the core of conceptual modelling and simulation model implementation in Part II. Results of this work have been published in several papers and articles, contributions by the author among others in (Lotzmann et al., 2015) and (Lotzmann and Neumann, 2017). Further references and project deliverables are mentioned below.

The chapter is structured as follows: After introducing the GLODERS research project, the rough outline of the customised modelling process is sketched, to be further specialised and filled with concrete sub-processes in the subsequent section.

4.2 The GLODERS project

GLODERS[40] — the acronym for Global Dynamics of Extortion Racket Systems — was a research project which started in October 2012 and ended in September 2015. It was co-funded by the European Commission in the Framework Programme 7 with a theme in ICT for Science of Global Systems, and brought together four partners from universities and research institutes in the UK, Italy and Germany.

In the project, a particular type of criminal organisations was investigated: Extortion Racket Systems (ERS), covertly acting organisations in various manifestations and with different goals spread over all parts of the world, with the common property of using extortion as important means to achieve their goals. Such organisations can be deeply entrenched in the legal society, or they can be just of peripheral matter. The most prominent example of the first kind (and for ERS in general) is the Mafia, in particular the Sicilian branch Cosa Nostra. An example for the second kind is a gang of drug dealers for which extortion is mainly a method to exert power among gang members.

Both types of ERS's have been investigated in GLODERS applying the most prominent research methods of Computational Social Science: data analysis and agent-based simulation (see section 2.3.1). Main result of the project are several simulation models covering different aspects of such criminal organisations:

- An ERS model implemented in Java (Nardin et al., 2016b) and Netlogo (Troitzsch, 2016b) (both replicating features of each other), simulating the interactions of the Cosa Nostra with the legal society in Sicily and investigating the effects of public movements against the Mafia (like 'Addiopizzo'[41]).

- A Mafia War model, dealing with "conflict escalation in criminal organizations, investigating conditions of stability and collapse" within the Cosa Nostra (Neumann et al., 2017).

- A model of a gang of drug dealers in a Western European country which experienced a violent collapse (spawning several cases of extortion) due to failure of internal conflict regulation mechanisms. This model will be elaborated in Part II of this PhD thesis, as it serves as the use case example for demonstrating the modelling process developed in OCOPOMO (see previous chapter) and reshaped in GLODERS (as this chapter will reveal).

All modelling activities in GLODERS are (at least loosely) based upon a common cognitive theory: a normative agent architecture called EMIL-A. The EMIL-A framework was developed for and tested with stylised-fact examples in

[40] http://www.gloders.eu
[41] http://www.addiopizzo.org

the previous project EMIL[42] (Conte et al., 2013). GLODERS elevated EMIL-A to a practical layer by using it in models of complex social systems with relation to real organisations and practical questions (Nardin et al., 2016a).

A thorough description of the GLODERS project can be found e.g. in (Elsenbroich et al., 2016). In the following, a distinct aspect will be introduced: the modelling process used for designing and implementing the third model in the list above.

4.3 Adapting the OCOPOMO process for data analysis: The GLODERS process

After deciding the use cases to investigate in GLODERS and examining the information and material available for creating the simulation models, it became evident that a modelling process as developed in OCOPOMO could be beneficial at least for the case of the collapse of a gang of drug dealers. The data base that should constitute the evidence for this model consisted of unstructured texts, stakeholders were to be involved in the modelling process and should be enabled to keep track of these activities, as well as to recognise their investigation results in some way or other in the simulation model and the presentation of results. Hence, the CCD and traceability concepts of OCOPOMO promised to be the ideal approaches to match the procedural requirements.

Since (fixed) information from completed criminal investigations had to be analysed as starting point for modelling the scenario method of the OCOPOMO process was not applicable. In the proposed approach it had to be replaced with appropriate data analysis methods due to the amount (several hundreds of pages) and the sort (completely unstructured) of respective texts. Summarising, the following areas of requirements for the concrete modelling tasks of the GLODERS project were considered (Lotzmann et al., 2015):

- stakeholder participation: stakeholders from police provide investigation files, empirical domain knowledge and discuss the state of development with the modellers on various occasions;
- type of empirical data: reports from criminal investigations (interrogation reports, court files).

As a first task the data analysis and simulation modelling team of the project[43] developed an outline for a combined qualitative and qualitative data analysis embedded in a conceptual modelling process. In Figure 4.1 this general idea for a modelling process is introduced, catering for the aforementioned requirements. Similar to the OCOPOMO process as sketched in Figure 3.1

[42] `http://emil.istc.cnr.it`

[43] For this particular use case model based at the University of Koblenz, Germany, in collaboration with the modelling team of CNR, Rome, Italy.

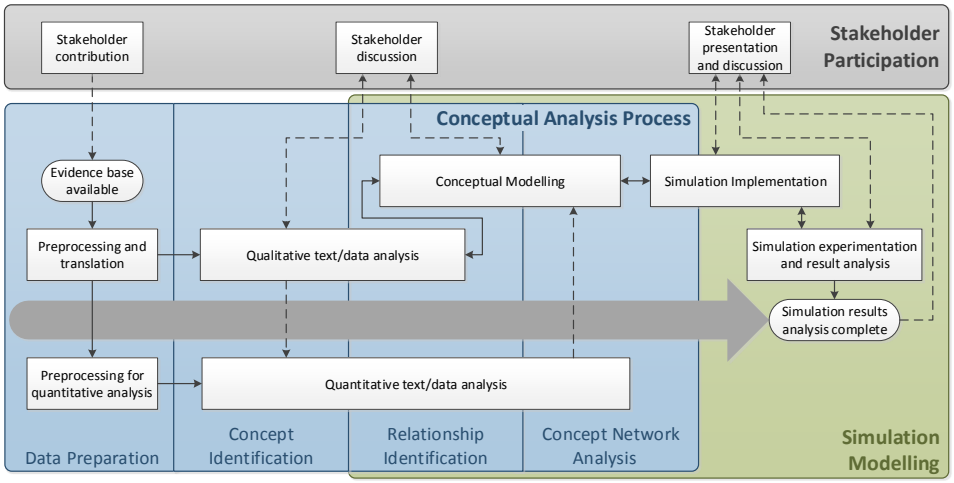

Figure 4.1: Overview of the evidence-based modelling process (background boxes represent aspects of the process; the flowchart highlights the activities, with solid lines showing the process flow and dashed lines depicting influential relationships)

the aim is to transform evidence data into a formal model and ultimately simulation results. As a significant difference, the property of having multiple iterations along the entire process is not regarded here, since it seems less important in the context of analysing already closed criminal investigations to regard the case that new insights would require to completely rethink the model. Instead the iterations within the data analysis and modelling stages are of very high importance, emphasising the necessity for a thorough and interwoven data analysis and modelling approach.

As point of departure the availability of an evidence base provided by the police stakeholders is defined, setting off a four-staged conceptual analysis process, operationalised from a concept first published in (Krukow et al., 2014, pp. 7-17). At the first stage a preprocessing (including a translation on the natural language level, if necessary) of the data is considered, in order to make the data processible for data analysis.

The envisaged kind of data — basically unstructured texts — and the sort of information to be extracted from the data — not only about structure of and relations between initially unknown concepts, but also manifold and complex descriptions of behaviour — pointed to a prominent role of qualitative analysis methods. Hence the process is defined around qualitative text analysis interlinked with conceptual modelling. Focus of the data analysis is the identification of concepts and relationships, while during conceptual modelling these relationships are further assessed and brought into concept networks. These two steps are performed in iterations, always having stakeholders on

board to discuss the validity of the developed concept.

Nonetheless, the inclusion of quantitative data analysis methods is considered here as well (Krukow et al., 2014). Applying relevant computerised methods (like text mining methods such as tokenisation and stemming (Aggarwal and Zhai, 2012)) on the texts makes further data preparation steps necessary, for example manual clean-up and the provisioning of a dictionary (words with similar meaning, spelling variants etc.). These additional inputs may also come from the qualitative analysis, and the results of the quantitative analysis feeds back into the qualitative and conceptual work.

The conceptual modelling is, on the other hand, linked with the simulation modelling phase as the model implementation receives its grounds from the conceptual model, and the formalisation might also reveal new insights for the conceptual modelling (for example the discovery of gaps — missing details to achieve a proper formalisation). Model implementation goes hand-in-hand with simulation experimentation, firstly to verify the model, later on with the aim to validate model assumptions and finally to generate simulation results for final analysis. These stages can again involve iterations, and the stakeholders have to be kept informed on the development progress and results.

The process finally applied for the actual use case model development was trimmed to qualitative data analysis[44]; it will be presented in the following section.

4.4 Building up the GLODERS process

The data analysis for the actual case started according to the process described in the previous section with data preparation and the initial stages of qualitative analysis. In discussions with stakeholders it was decided to concentrate on aspects that were dependent on qualitative analysis, giving the quantitative side less priority. In the course of the project it became apparent that a sophisticated methodology for qualitative data analysis combined with conceptual modelling was a substantive contribution on its own. Neumann formulates this in (Lotzmann et al., 2015) as follows:

> It "is a demonstration and proof-of-concept of the approach to making use of evidence based modelling as a core tool in a process of grounded theory development (Neumann and Lotzmann (2016a); Dilaver (2015)). Central to this account is to rigorously grounding model assumption in empirical narratives (Lotzmann and Wimmer

[44]Actually a quantitative analysis of the data also was performed in GLODERS in order to reveal money laundering networks. Starting with AutoMap for identifying concepts and finding co-occurrences, these results were imported into ORA for a measurement-based network analysis. Restricting the model to specific aspects (as highlighted in section 5.3) rendered these additional details about money laundering activities irrelevant. Details of this quantitative analysis are given in (Sartor, 2015); for information on the tools used, see `http://www.casos.cs.cmu.edu/tools/`.

> (2013b); Lotzmann and Wimmer (2013a)). Thus it takes a stance in the ongoing debate on simple or realistic modelling strategies (Coen, 2009). Evidence-based modelling is an approach of developing models not on simple theoretical assumptions but on factual empirical evidence, sometimes referred to as KIDS principle (Edmonds and Moss, 2004) by making use of ethnographic accounts to get away from numbers (Yang and Gilbert, 2008). This is particularly popular in participatory modelling (Barreteau, 2003). Its growing recognition is demonstrated in a recent special issue on using qualitative evidence to inform the specification of Agent-Based Models in Vol. 18(1) of JASSS[45]."

Hence, the further research on this 'restricted' approach set grounds for a self-contained process. This section will give a description on the relevant details as published in (Lotzmann et al., 2015), which will then be illustrated in the next chapter (Chapter 5) by a concrete use case example.

This process also includes an additional requirement from GLODERS: involving normative agents based on an existing theoretical foundation to enhance the model design. Hence, the design of the process has to consider a number of different perspectives: The conceptual analysis process, the simulation modelling and the stakeholder participation as already included in the generalised process, plus an additional theory of normative agents. These perspectives are shown in the background lanes of Figure 4.2.

The first aspect, the model development as core of the process, is likewise divided into sub-processes for conceptual analysis and simulation modelling, which are overlapping and exhibit temporal dependencies. Starting with the availability of documents in the evidence base, the analysis is comprised of the following four steps (Krukow et al., 2014):

- Data preparation for ordering and standardising the textual data for further processing. This includes unifying the data from different file formats, data encoding and language translation.
- Concept identification, i.e. the (basically manual) extraction of meaningful words and phrases.
- Relationship identification, which starts with a categorisation of the identified concepts, followed by an extraction of relationships between the categories.
- Concept network analysis, based on the identified relationships, e.g. by appropriate visualisations.

The first design steps for simulation modelling can be started already when the preliminary results from the relationship identification phase are available, the

[45] http://jasss.soc.surrey.ac.uk/18/1/18.html

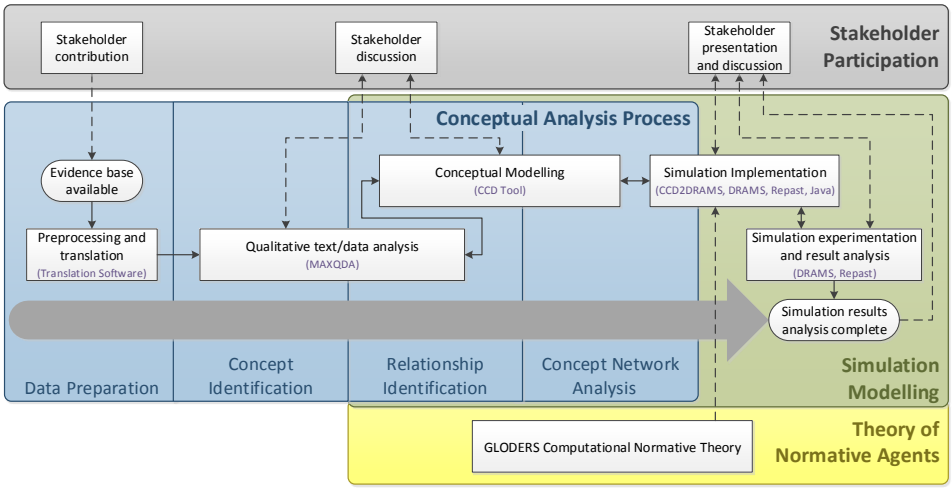

Figure 4.2: Overview of the evidence-based modelling process (with GLODERS-specific details) (background boxes represent aspects of the process; the flowchart highlights the activities, with solid lines showing the process flow and dashed lines depicting influential relationships)

actual simulation modelling takes place after the conceptual analysis process, and eventually leads to validated and analysed simulation results.

The second aspect is a theory of normative agents which drives the entire analysis and modelling process. A normative agent is capable of normative reasoning, i.e. to memorise (social or legal) norms and use them for decision making. Foundations of the applied theory were originally developed in previous research conducted in the EMIL project (Conte et al., 2013), and extended and concretised within GLODERS (Andrighetto et al., 2013).

The third aspect regards the practitioners' perspective of the stakeholders who provide the empirical domain knowledge and have a significant interest in the activities and results of the different stages of the development process.

The process as it has been applied in the model development is sketched in Figure 4.2. Like the general process in Figure 4.1, details of the specific activities carried out during the development are given for the particular use case, but in addition the applied tools are highlighted.

The initial activity is the translation of the evidence base consisting of police interrogation protocols into English language using translation software, followed by a manual correction of errors caused by OCR and translation using text processing tools. This step can be simplified or even omitted in other cases, if the original documents are available in electronic formats (i.e. no scan and OCR process needed, which is highly preferable). With the texts prepared, the conceptual analysis process can be performed by a qualitative

text analysis in the sense of a grounded theory approach (Glaser and Strauss (1967); Corbin and Strauss (2008)), using the MAXQDA[46] software, a standard tool for Computer Assisted Qualitative Data Analysis (CAQDAS). The results from this analysis are transferred into a Consistent Conceptual Description (CCD), which contains concept representations of the entities relevant for the simulation, like actors, (mental and physical) objects, the relations between both kinds of entities, and — most importantly for this stage of modelling — the description of the dynamics constituted by the actors' behaviour by means of condition-action sequences (or action diagrams, according to the CCD terminology) (Neumann and Lotzmann, 2016a).

As mentioned before, the process of qualitative analysis is heavily interwoven with the conceptual modelling; insights from both activities influence each other manifoldly. In particular the annotation feature of the CCD (for enabling empirical traceability from model elements back to the evidence) fertilises the analysis process. Also stakeholders can and should be involved in these two activities.

The CCD is then used to develop the simulation model, starting with a model-to-code transformation (CCD2DRAMS) and subsequent implementation as a declarative rule-based model in DRAMS and Java, using an agent architecture derived from the GLODERS computational normative theory (Nardin et al., 2016a), and running in a Repast (North et al., 2006) environment. Simulation experiments are performed, firstly for verification and validation purposes, afterwards in productive runs generating results to be analysed and presented to the stakeholders. The results are expected to also shed new light on aspects of the normative theory, like sanction recognition mechanisms.

In the following subsections the different phases — qualitative analysis, conceptual modelling, simulation modelling and experimentation — will be presented in more detail.

4.4.1 Qualitative analysis and conceptual modelling

Figure 4.3 shows the qualitative analysis and associated conceptual modelling step-by-step.

As comprehensively described in (Neumann and Lotzmann, 2016a), after the initial data preparation, a MAXQDA-based analysis is performed. Relevant text phrases for the central parts of the model — in this case the collapse of the criminal network — are identified, and concepts (recorded as codes) are classified, i.e. codings are developed. In conjunction with these activities the concepts are annotated with related text phrases, resulting in a preliminary concept model. Figure 4.4 shows the user interface representation (as provided by MAXQDA) of the list of codes for which the relevant coding (with annotated text phrases) can be put into focus, together with the associated annotations

[46] `http://www.maxqda.de`

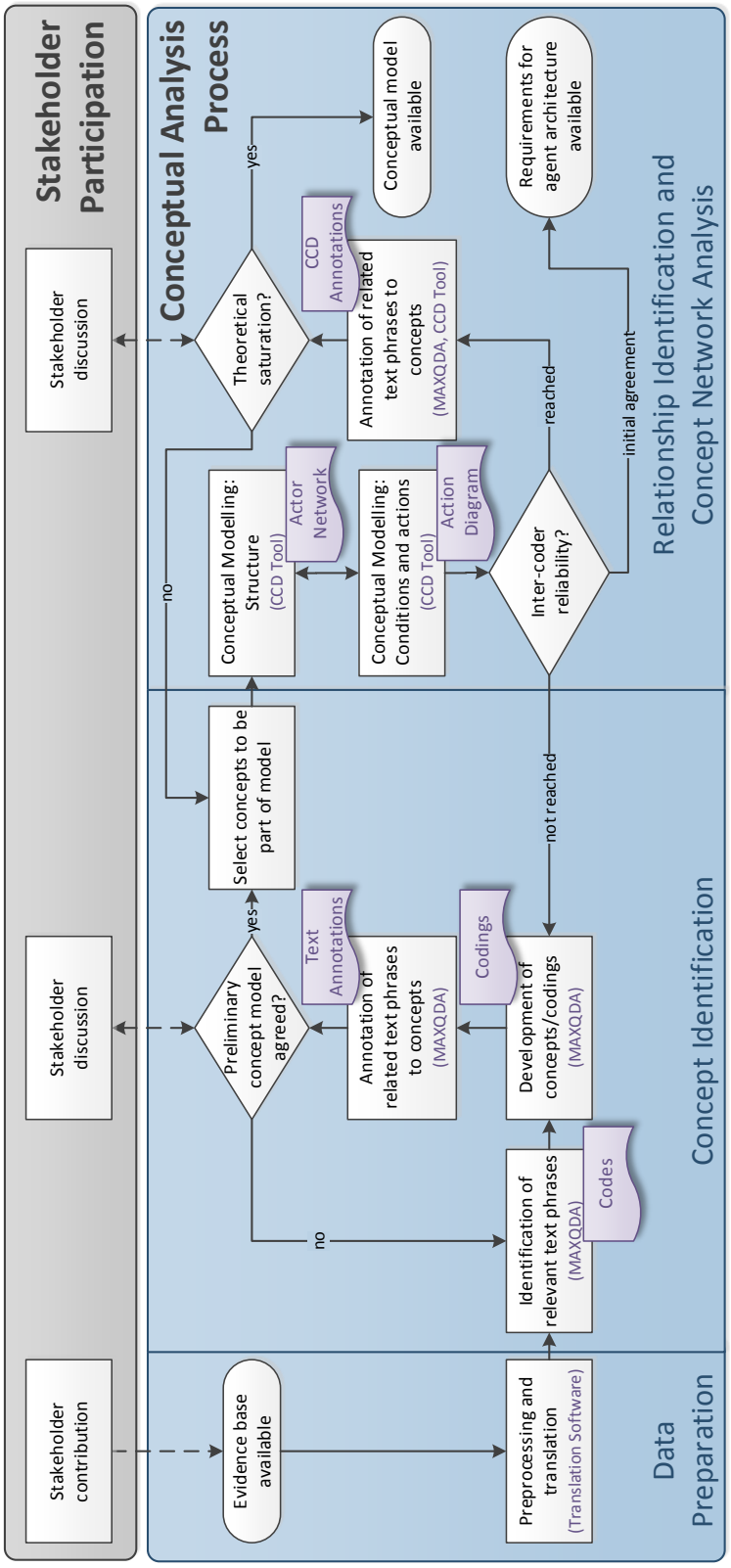

Figure 4.3: Details of the qualitative analysis and conceptual modelling process

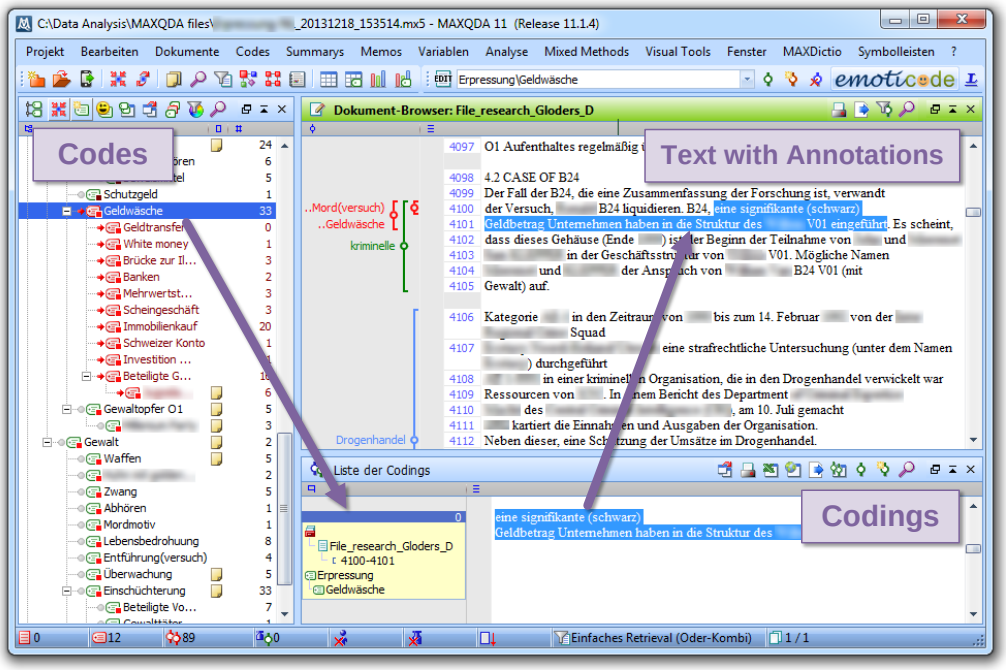

Figure 4.4: MAXQDA user interface with codes, codings and annotated text

situated in the original text. The concept model needs to be discussed with the stakeholders in order to ensure the empirical and practical relevance of the concepts. This discussion typically leads to a revision of the concept model, based on newly identified relevant text phrases. Thus, the stakeholder knowledge needs to be exploited to identify the most comprehensive set of concepts.

When the concept model is finally agreed, in a discussion between analysts and modellers (in a collaborative approach, as performed in GLODERS), concepts relevant for (and realisable in) a simulation model are identified and recorded in a CCD by using the CCD Tool of the OCOPOMO toolbox, firstly as a condition-action diagram (dynamic aspects, Neumann (2014); for an example, see Figure 5.11 in Chapter 5), followed by the development of an actor-network diagram (static aspects; Figure 5.3 in Chapter 5 shows an example). This conceptual model is compared with the evidence and becomes subject of the discussion in order to achieve a state of inter-coder reliability[47], which requires several iterations of the involved process steps (i.e. development of new concepts/codings and incorporate these into the CCD). Already at this time the first 'technical' developments are possible (like for this particular model

[47]The term inter-coder reliability (ICR) is used here in more general terms as defined e.g. in (Lombard et al., 2010) as "the extent to which independent coders evaluate a characteristic of a message or artifact and reach the same conclusion". Statistical methods to calculate values for ICR as suggested in (MacPhail et al., 2016) have not been taken into consideration.

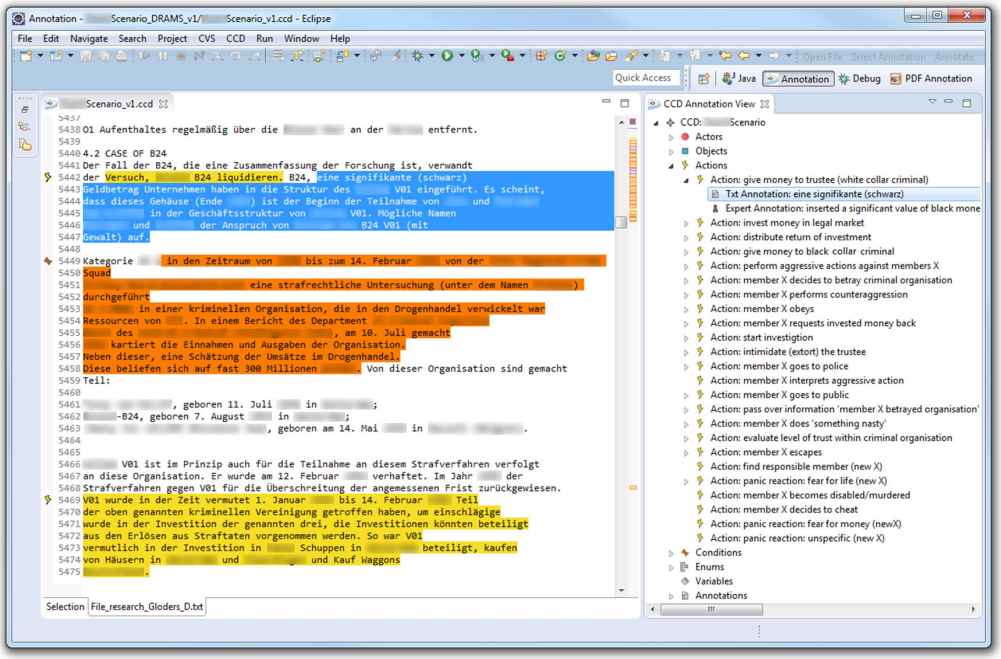

Figure 4.5: CCD tree view with text annotations

the normative agent architecture), since the first concrete requirements for the implementation can been extracted from the conceptual model developed so far.

Once the inter-coder reliability is achieved, the CCD content is enriched with annotations of the related text phrases and MAXQDA codings; Figure 4.5 gives a screenshot of the CCD user interface displaying the CCD on the right hand side and the evidence text with (coloured) annotations on the left-hand side. This procedure of enriching the CCD is done until a level of theoretical saturation (Corbin and Strauss, 2008) is reached. This can again be considered a recursive process, as (Lewis-Beck et al., 2004, p. 1123) define theoretical saturation as "the phase of qualitative data analysis in which the researcher has continued sampling and analyzing data until no new data appear and all concepts in the theory are well-developed".

Finally, the validity is ensured by consulting the stakeholders once again, after which the actual process of implementing the simulation model is initiated.

4.4.2 Simulation modelling and testing

The process of concept analysis of empirical data described in the previous section provides the basis for the process of developing the simulation model, as shown in Figure 4.6. It is assumed here that the simulation model is imple-

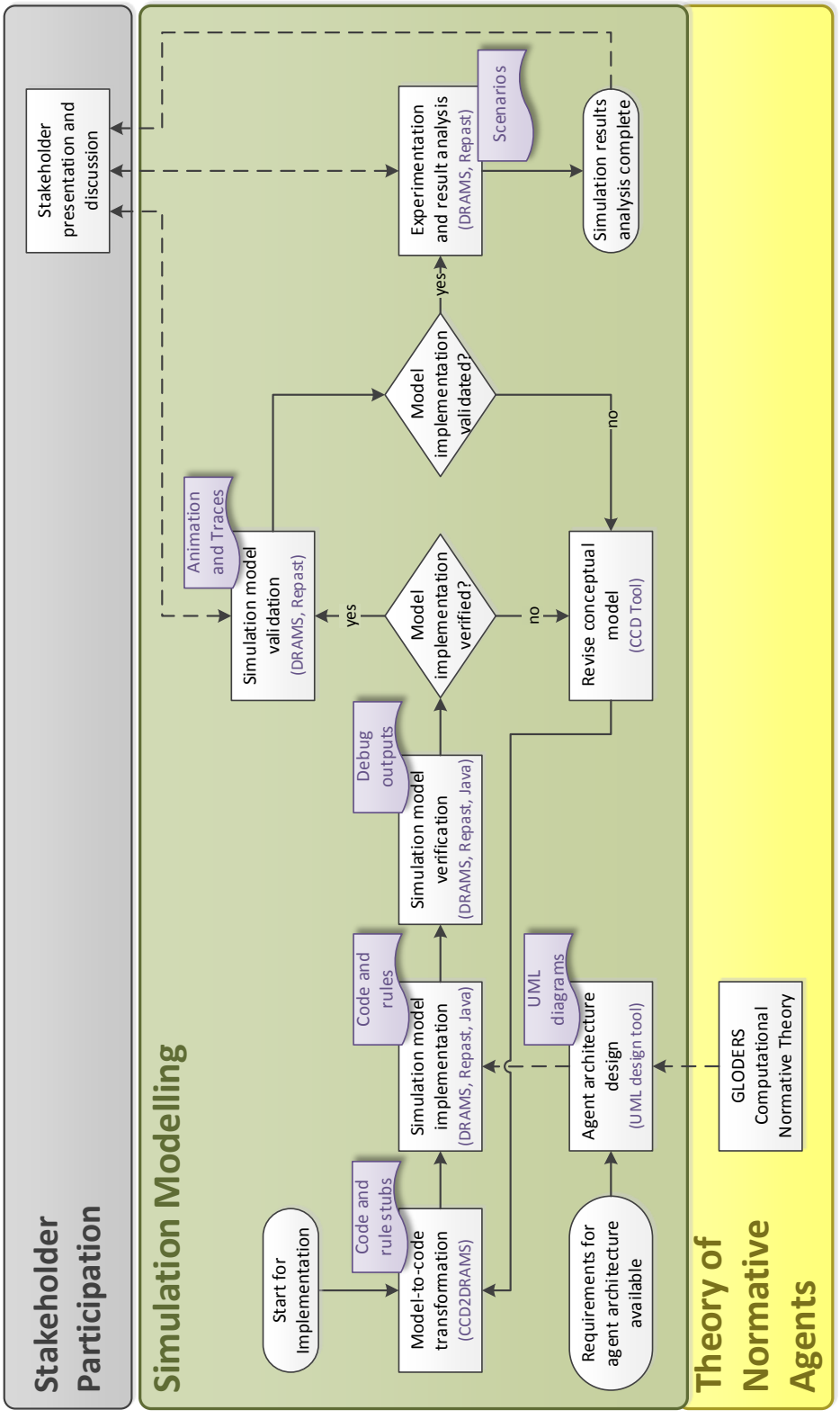

Figure 4.6: Details of the simulation modelling process

mented in the way proposed by OCOPOMO and introduced in section 3.3.4, i.e. using DRAMS together with Repast as technological basis. However, this process step is further detailed for the GLODERS application.

As a first step, the CCD2DRAMS tool is used to transform the CCD into a skeleton of simulation model code. These generated rule stubs are then elaborated, i.e. completed with valid program code. This implementation also has to take other aspects into account, such as the agent architecture previously designed on the base of requirements that have arisen from the conceptual modelling or the normative architecture. These aspects are conceptualised, for example in the form of flow charts — an example is given in Figure 5.9 in Chapter 5.

The implementation, too, is part of the aforementioned process iterations, and associated with persistent testing. Gaps identified in testing usually have effects on the code, but occasionally also the CCD is affected when certain crucial model aspects (such as parameters) appear inconsistent or are missing. After the model is successfully verified, a validation takes place, involving the stakeholders in discussions about the preliminary results. Here once more the need for modifications might become apparent, which then would trigger a further adaptation or extension of both the conceptual as well as the formal model. Finally, with the validated model 'productive' experiments generate results in form of logs of scenarios. These are fed into subsequent analysis and interpretation. The final results are scenarios formulated as 'short stories' with the aim to give feedback to the stakeholders in a language appropriate to the application domain — in the case of criminology some sort of 'crime novels'.

The DRAMS simulation model implemented in this process ideally follows closely the representation of dynamic aspects in the CCD action diagram: If certain conditions are satisfied, here expressed by the availability of certain facts stored in fact bases, a number of actions can take place, here by triggering certain agent rules. Also code catering for other requirements can be realised by declarative rules (as for the normative aspects in this case).

In order to complement the screenshot in Figure 3.9 in section 3.3.4, a more elaborated example of a code fragment taken from a simulation model is given in Figure 4.7. It shows a declarative DRAMS rule (`defrule` construct) in the Eclipse-based development environment. Again, the comments above the rule definition are automatically generated by CCD2DRAMS and contain copies of the annotations associated to the related CCD-action, together with a reference UUID (`@link`) to this CCD element. Here, in contrast, the rule is already fully implemented.

According to the fundamental idea of the CCD, an implementation should not only follow the structure of the CCD action diagram, but also — as far as possible — the other elements and constraints given by the conceptual model. This concerns e.g. the objects defined in the CCD, which should be incorporated into model code as data elements to be manipulated by the agents.

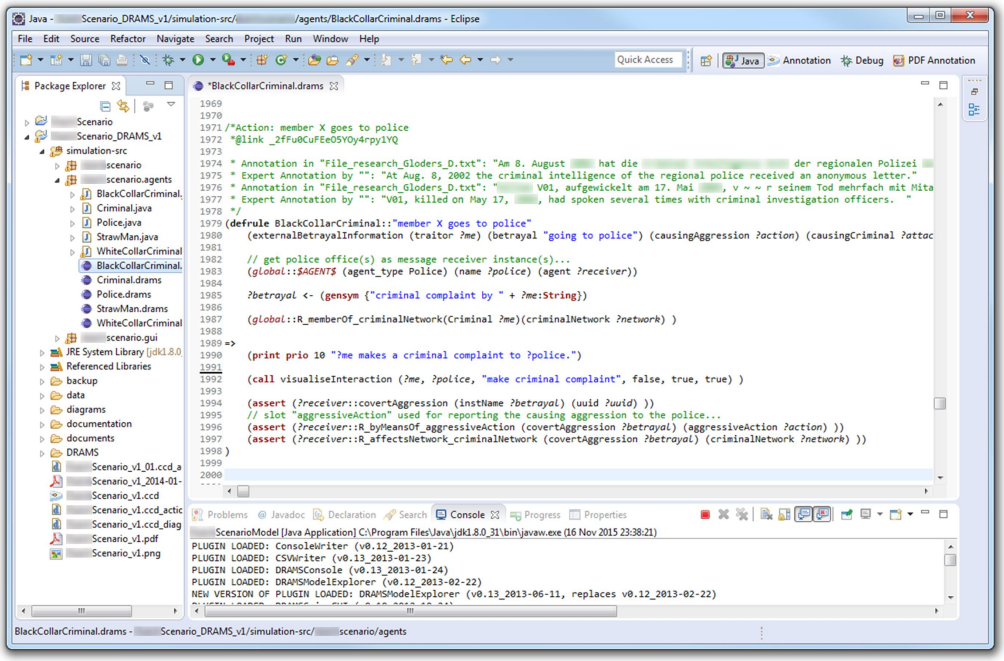

Figure 4.7: Example of DRAMS rule with annotations derived from CCD and generated by transformation tool CCD2DRAMS

However, in this particular case where the purpose of the conceptual model is more concerned with enabling communication with involved people like data analysis experts and stakeholders from the application field, the complexity of such CCDs might not be sufficient for a reasonable model structure. The CCD2DRAMS tool can only create code with the complexity of the CCD.

Although it would be possible to 'squeeze' the additional code needed for an executable simulation model in just a small number of rules, this is not a recommendable approach: it results in rules consisting of extensive, unstructured and, thus, unreadable code. In effect, just relying on the generated code elements will make the model cumbersome and hard to maintain. Hence, additional rules and fact templates should sensibly be defined. For example, a best-practice (similar to object-oriented programming) is to dedicate an individual rule for every distinct concern, keeping the size (mainly in terms of 'lines per rule') at a manageable level (no more than twenty as a rule of thumb). Also, not all tasks can be efficiently coded in declarative rules. For example extensive calculations, character string manipulations or special runtime visualisations should be 'outsourced' to imperative code, where Java methods are the means of choice. DRAMS supports the dynamic inclusion of Java code in several convenient ways.

In particular for these cases where the complexity of the rule structure exceed the complexity of the CCD action diagram, the code visualisation func-

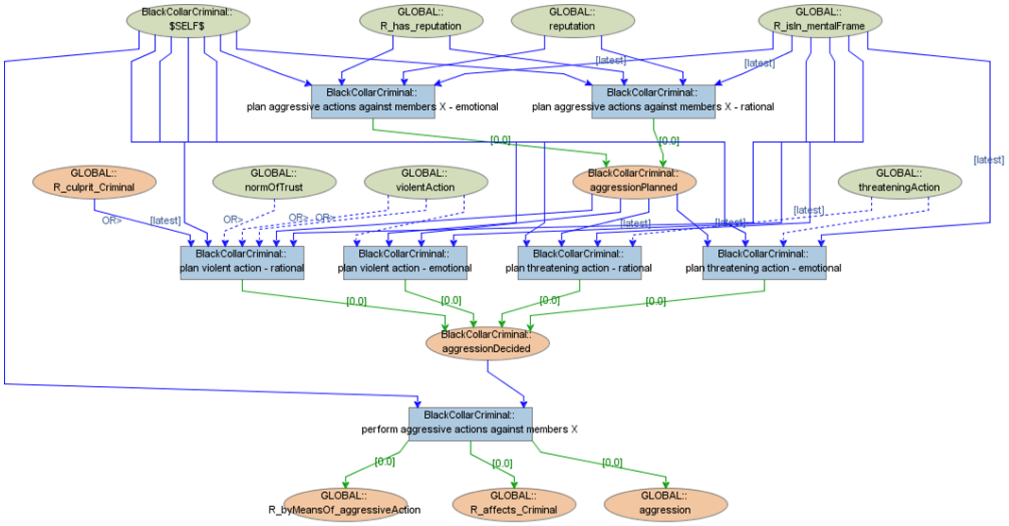

Figure 4.8: Example of Data Dependency Graph showing the rules and facts involved in a multi-stage decision process

tionality of DRAMS becomes a crucial feature in order to keep track of the implementation. The dependencies between implemented code constructs (rules, fact templates and facts) are drawn in the Data Dependency Graph (DDG). Figure 4.8 gives another (and more complex as in Figure 3.11) example of such DDG, reflecting just a single action of the CCD action diagram while covering a multi-staged decision process: An agent representing a criminal has to reason about the kind of an aggressive action to be performed against another agent. The role of the DDG and the technical background for this representation are detailed in Chapter 7 and Part III, respectively.

4.4.3 Simulation experimentation

In order to run experiments with the simulation model, parameters have to be set according to experiment designs, which are formulated on base of questions to be answered by simulations.

There are typically many different ways to implement parameters in models: either hard-coded in model code, accessible via (external) variables, or — as in this case — even outside the simulation model in the CCD. Some parameters — especially regarding the model structure including types of agents with certain attributes, types of aggressive actions etc. — should be configured in the CCD, mainly to benefit from the traceability feature of the OCOPOMO toolbox. The CCD allows to define instances for actors, objects and relations, that constitute the simulation world. In contrast, parameters such as number of agents or probabilities for certain decision options are set by parameter files

or via dedicated user interfaces.

Running simulations is controlled by the user interface of the simulation tool and generates various kinds of outcomes. Besides visualising some numerical output variables with e.g. time series diagrams (or applying statistical methods on these variables), the results of simulation runs are presented as simulation logs, which basically tell 'stories' of the events in a human-readable way.

The experiments to be conducted with the model have to be designed in a way that the stakeholders' issues are addressed, i.e. relevant questions of the stakeholders are treated. In order to find answers to such questions, traceability is a valuable instrument. A currently available analysis tool is the experimental Model Explorer, a plug-in for DRAMS capable to visualise these traces. This software is primarily dedicated for model debugging and code verification, but also useful for validation purposes. This topic is examined further from a technical perspective in Chapter 10. The simulation result analysis and the applied tools are demonstrated by means of the use case model, which is subject of Chapter 7.

4.5 Conclusions from the GLODERS project

This chapter showed how the OCOPOMO process introduced in the previous chapter is customised and applied for a use case in the application area of criminology within the GLODERS project. The OCOPOMO model development process can be reused nearly unaltered, after replacing the scenario method for evidence collection by data analysis techniques. Apart from that, the other main contribution of GLODERS is the detailing of the different process steps, tailored for the special need of the use case on the one hand, but on the other hand is general enough to be applied also for other applications. These can be applications which fulfil (at least a subset of) the following criteria:

- evidence based models, with (unstructured) texts as raw material;
- ideally involving different roles in the modelling endeavour: data analyst, modelling expert, domain experts/stakeholder;
- use conceptual models for communication purposes, i.e. as a means to establish a common language between the domains, and between the persons (or roles) belonging to these domains;
- traceability is a relevant requirement; in this case, an appropriate tool support is crucial (such as provided by the OCOPOMO toolbox).

Part II of this PhD thesis thoroughly discusses the aforementioned use case example, for which — of course — all these criteria match. In summary, the insights obtained by performing the work in GLODERS (and in particular on the use case model) revealed a great potential impact regarding quality and

trustworthiness of models by stakeholders and non-simulation (but domain) experts.

Extensions of the work done in the GLODERS project (besides the points already made in the conclusion section of the previous chapter) could include the following aspects (Lotzmann et al., 2015):

- Open up the process to a broader range of methods and techniques, in particular with respect to more sophisticated simulation result analysis procedures which entail the full impact of the traceability means.
- Quantitative data analysis methods could be included on a more detailed level, as already foreseen in the general 'GLODERS process'. This could be achieved among others by providing elaborated interfaces to such methods and tools, or more generally
- a tighter integration of the different software products used at the single process steps should be targeted in an integrated toolbox.

Part II

Application

Chapter 5

Use case: An application in criminology

5.1 Introduction

This chapter provides the 'entry point' to the first practical part of the thesis — the conceptualisation, development, execution and result analysis of a simulation model of intra-organisational dynamics of a criminal network. The results presented in Part I are the methodological frame for this Part II.

In this Chapter 5, the conceptualisation phase of the modelling process is covered. Parts of the content presented here have been previously published, most importantly in (Neumann and Lotzmann, 2016c). Other references are included below.

The chapter is structured as follows: Firstly, the modelling project is motivated, taking up the discussion of section 2.4. Afterwards the real-world subject is introduced, followed by a detailed conceptualisation of the relevant aspects, where also the relations to the preceding data analysis phase are pointed out. The final sections are then dedicated to pave the way towards formalisation (and ultimately implementation) of the model, including an overview on the respective requirements.

A note on reading this chapter: The conceptualisation of the model is done step-by-step, whereby in the individual sections fragments of the model are introduced. Starting with the different facets of behaviour of the criminal actors in distinct situations in section 5.4, the underlying static structures are presented afterwards in section 5.5.1. The key to assemble these different and seemingly artificial views into a coherent picture is section 5.5.2, where instances belonging to the concepts are illustrated.

Another note of the text style policy used in this chapter: `Typewriter style` is used for identifiers of entities shown in diagrams — mainly used in the context of action diagrams. I contrast, *slanted style* is used for text phrases directly referring to elements in diagrams, without necessarily using the exact name of the elements.

5.2 Motivation

This Part II of the PhD thesis will demonstrate the practical usability of the evidence-driven modelling process described in Part I to design and ultimately implement, verify and validate a simulation model with the tools presented in Part III. This chapter introduces the application case and shows how this case is conceptualised. In the subsequent two chapters the actual implementation and the analysis and presentation of simulation results, respectively, are elaborated.

The application field selected for this modelling example is criminology: the violent collapse of a network of drug dealers in a Western European country. Unlike many other examples — such as presented in (Elsenbroich, 2017), (Nardin et al., 2016b), (Troitzsch, 2016b), (Malleson et al., 2013) or (Bosse and Gerritsen, 2008), see also section 2.4 — not the interaction of criminal organisations interacting with their legal environment is in focus, but the inside perspective within the 'illegal world'. Motivations for selecting this example are threefold:

1. This topic was selected (upon stakeholder initiative) as one of the use cases within the GLODERS project.

2. This modelling example allowed to bring together experiences from previous projects EMIL and OCOPOMO with GLODERS. While EMIL contributed a normative theory which aimed at equipping agents with rich cognitive capabilities, in OCOPOMO the grounds for a structured participatory approach to model development, conceptual modelling and traceability of evidence data were set (Chapter 3). GLODERS used the assets of the two projects and extended these with data analysis techniques (Chapter 4).

3. The size of this model is sufficiently beyond trivial example models, but still small enough to remain comprehensible in the context of this PhD thesis.

With view on the story of this criminal network, the overall research question can easily be formulated as follows:

> How could this escalation of violence happen?[48]

Other than that, several different perspectives and associated research questions were regarded for this simulation modelling project. Firstly, the stakeholder from police who provided the substantial material for the evidence base approached the modellers with two additional questions:

[48] "Wie konnte es zu der Gewalteskalation kommen?" (Neumann and Lotzmann, 2016a, p. 279)

> Is the evidence provided rich enough to inform formal modelling and simulation?
>
> From a criminological viewpoint, can new insights and results beyond the already concluded 'classical' police investigation about the case be achieved from simulation? (Neumann and Lotzmann, 2016c)

Secondly, from the perspective of social sciences a number of relevant research questions are detailed in (Neumann and Lotzmann, 2016a), which can be sketched as a brief summary:

> Is it possible to gain insights on the fundamental social-theoretical question of conditions for social order from analyses on a microscopic (i.e. individual) level?[49]

The third perspective is the one from computer science, and in particular its relation to social sciences. This is the perspective focussed on in this chapter, complementing the two other perspectives that are specifically focussed in the publications mentioned above. The following research questions are of interest in this context:

> Is it possible to apply a stringent modelling process inspired from software engineering for the realisation of complex simulation models which can be regarded as a special class of software systems with typically soft and blur requirements, while these requirements are not only biased through stakeholder engagement (which is the baseline e.g. for industrial applications), but also during the process of conceptual modelling, implementation and execution of simulations, triggered by verification and validation of the developed software, and in addition by theories of human behaviour?
>
> Does thorough and comprehensive software specification result in added value regarding quality of both model implementation and simulation results?
>
> Does traceability from simulation results back to evidence documents increase the acceptance of the simulation method for stakeholders?

As already mentioned, files from police investigations provided by a police officer who took part as stakeholder in the GLODERS project served as starting

[49] "Die mikroskopische Analyse erlaubt jedoch Einsichten in grundlegende Fragen der Sozialtheorie nach den Bedingungen sozialer Ordnung [...]." (Neumann and Lotzmann, 2016a, p. 279)

point for the modelling tasks. The files are partly confidential and contain mainly interrogation protocols of witnesses and defendants, collected in several investigations which partly took place way after the actual collapse of the network. These protocols cover information about different aspects related to the criminal network concerned: Firstly, the routine of the everyday business of drug trafficking and money laundering, relations between people involved in the network and their role in the everyday business (e.g. 'regular' criminals, businessmen, stooges etc.). Secondly, the events involved in the course of collapse: from friendship, encapsulated mistrust, on to 'rule of terror'. Acts of aggression and violence, among them numerous cases of murder were recorded. This aspect of understanding the network-internal mechanisms that ultimately lead to the complete blow-up of the network sets the driving research question for the simulation model.

As mentioned earlier, scientific work in the criminal sector often focusses on the interface between legal and criminal world, since this is the obvious point of attack to fight crime. However, understanding the internal hidden dynamics of a criminal network or organisation doubtlessly helps to assess and re-evaluate the interaction between the two worlds. Hence, the aim of this work is to investigate if simulation can help to shed light into the opaque criminal world. An overview of the social system regarded here is shown in Figure 5.1.

The outer circle visualises the legal world as encompassing environment, in which the (hidden) criminal world is embedded in some ways. The criminal network can be seen as an (institutional) actor mainly of the criminal world, but has also connections to the legal world, most prominently as source of profit by activities summarised under the term 'everyday business'. This is treated as a black box here, with all the kinds of aggressions and violence involved in these activities. As this everyday business is based upon trust among the members of the criminal network, a condition of mistrust might impede these activities. Such mistrust may be provoked by internal disturbances within the everyday business itself, but also conditions set by external actors can cause mistrust: Interventions by the police or even information about the criminal network in the hands of the general public (both are regarded as representative actors of the legal world), but also external events triggered by unknown actors from the criminal or legal world can set such a condition. As a consequence of mistrust, an internal conflict within the network might arise or be fostered. The behaviour originating from this internal conflict on the one hand impedes the everyday business, on the other hand leads to some kinds of aggressions among the network members. Aggressions of violent nature (e.g. murder) might become visible beyond the borders of the criminal world. In any way, aggressive actions amplify the mistrust further. So, the basic mechanism is set for a vicious cycle of aggressions and the breakdown of trust.

In the next but one section this concept of an internal process compromising the stability of the criminal network is spread out in more detail and gets underpinned with relevant citations from the evidence base. Prior to this, the

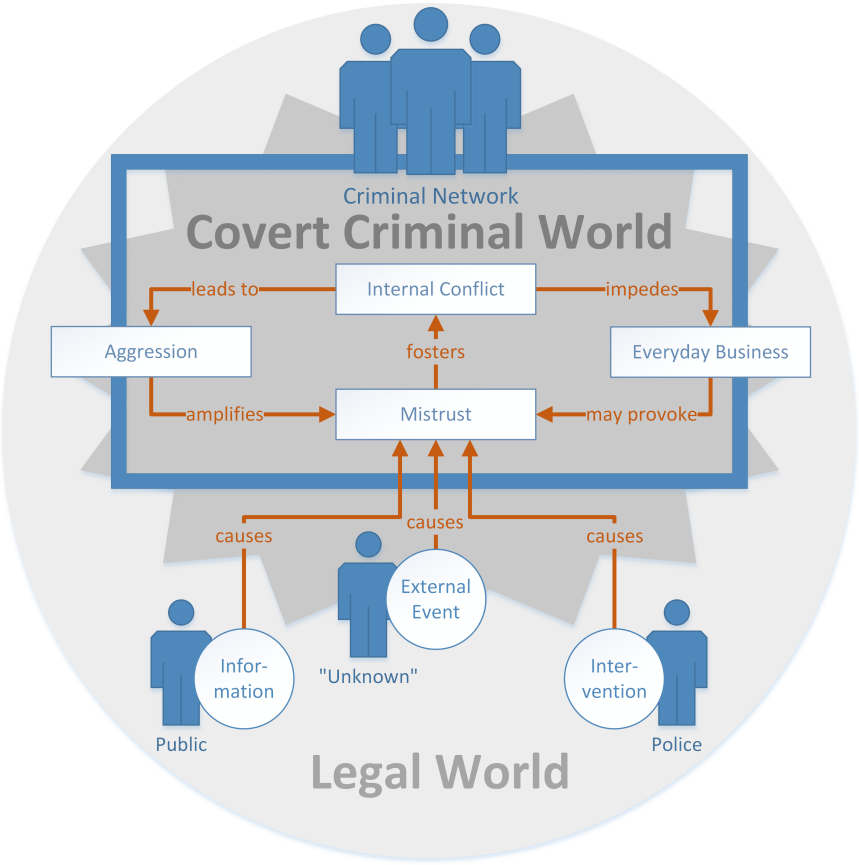

Figure 5.1: Overview of criminal network: internal dynamics and relation to legal world

story of the actual criminal network is outlined, though under the premise of keeping personal data of the persons involved protected.

5.3 The Story: Collapse of a criminal network

The criminal case modelled here on base of files from police investigations focusses on the fate of a gang of drug dealers that faced a sad demise after quite a long period of prosperous criminal business. The story of the criminal gang — as described in (Neumann and Lotzmann, 2016a) and (Neumann and Lotzmann, 2014) — started in the early 1990s with a couple of friends who knew each other from previous criminal ventures. Over the time the business with dealing drugs grew bigger and bigger, so that finally more than 200 people got involved in various ways and roles, while the core group consisted of about 20 to 30 people, most of them long-term friends. Within this core network no strict hierarchy nor any kind of (pre-) determined organisational structure was established, in contrast to other criminal organisations like e.g. the Mafia. The motivation of the network members was to jointly use the opportunity to

'make money' out of criminal acts, which they did with great success: Several hundreds of millions of an European currency were earned with drug trafficking. Some of the members had extremely high incomes, which resulted in extravagant lifestyle, including for example own private jets.

The huge turnover of the criminal network made proper 'professional' money laundering procedures a crucial prerequisite for the sustainability of business. This could only be achieved by involving people in the criminal network who were not only trustworthy for the core members, but also possessed reputation in the legal business world. This role was taken by a highly reputable business man with connections (among others) to construction financing enterprises, so that the money laundering could be performed by investment in construction projects e.g. at a European airport. Hence, this businessman (in the following called the White Collar criminal) was in his role as money launderer the trustee for the real, so called Black Collar criminals. The other people involved can be subsumed under the label 'Straw men' (and women), stooges with the tasks of backing and obfuscating criminal activities.

Despite the successful criminal business over a period of more than ten years, within two years the criminal network broke down — apparently fully unexpected — in a spiral of violence, murder attempts and also several successful homicides. The surviving former members of the network described this collapse afterwards in police interrogations with phrases like "corrupt chaos" and a "rule of terror" where "old friends were killing each other".

As it turned out later, neither the initial cause nor the stages of escalation of violence could be fully understood by the network members, so they had no chance to encapsulate the violence and stop the breakdown. Here the simulation steps in with the attempt to provide answers to the fateful question which could not be answered by the people involved in the network nor the police: How could this escalation of violence happen?

To cast the story of the collapse in a simulation model on base of data gained from police investigations, mostly interrogation protocols, raises questions about possible problems with this kind of data:

- Accused persons and witnesses might not remember the details correctly, or overemphasize less important events, or might not tell the truth.
- As in any investigation in covert areas of the society, there is the problem of dark figures. Events not concretely addressed in the interrogations are likely to remain invisible.

Therefore the involvement of police stakeholders in the modelling process plays an important role for clarifying vague or ambiguous assertions and for filling gaps in the data in a sensible way.

Details of the collapse and the related crimes will be described in the following sections where the analysis of the interrogation protocols and a conceptual model of the collapse is addressed. The subsequent chapters cover the simulation model (6) and the results obtained by simulation runs (7).

5.4 Conceptualising the case

Aim of this section is to outline the conceptual model of the above-mentioned case of criminal network, developed from the available police investigation material and stakeholder discussion. The conceptual model is not presented here in a purely result-oriented description, but rather looks at the process in which the conceptual model was developed. This is done in order to illustrate the modelling process introduced in the previous chapters by means of an example, and to give some insights in the thinking processes behind modelling steps and decisions. Presentations of the conceptual model concentrating on the results have been published already by Neumann and Lotzmann (Neumann and Lotzmann (2016a); Neumann and Lotzmann (2016c)), which are recommended for a quick reference on the topic.

An intense collaboration between a Data Analyst (Neumann) and a Simulation Modeller (Lotzmann) was performed over a period of several months. A series of sessions was conducted with the objective to identify the relevant aspects and to decide upon appropriate abstractions to be included in the model:

- The unknown trigger of the spiral of violence should be regarded as an external event with the consequence that a member of the network was reproached for a violation of the 'norm of trust'.

- The observed reactions of criminals on such a norm violation should be considered as punitive measures of criminal actors in the model.

- The observed reactions of criminals on any kind of aggression — be it punishment or for any other reason — should be considered.

- The consequences of any kind of aggression from the real case should also be considered.

The latter three issues were kept as close as possible to the real criminal network. As a 'key performance indicator' it was defined that the model should be able to reproduce the series of events which brought down the real network — as just one possible outcome, but being able to generate various other scenarios as well. All these aspects should be documented in a first draft of a consistent conceptual description (CCD).

In accordance with the modelling processes described in Part I, a consistent conceptual description (CCD) must involve consistent submodels for static aspects (represented by the actor-network diagram) as well as dynamic aspects (represented by the action diagram). During the collaboration the CCD was used as means for communication and documentation from a very early stage on. As the discussions at this time were focussed predominantly on 'dynam-

ics'[50], the action diagram of the CCD became pivotal element of the research. At the same time, the actor-network-diagram was synthesized in the background, just providing a scaffold of static elements to support the dynamic effects. In this respect the approach followed for this case deviated from other projects where this modelling process had been applied before (like in the OCOPOMO pilots — see Bicking et al. (2013)). There both the static and the dynamic aspects are regarded with equal (or even opposing) priority, since the employed scenarios and evidences provide a clear picture on both the static and the dynamic side. This proves the flexibility of the CCD approach to be adaptable to specific needs.

5.4.1 The Prologue

As a first step, both the Data Analyst and the Simulation Modeller needed to familiarise with the expertise and methodological background of one another. For this purpose, and as an exercise to evaluate the applicability of the CCD for the purpose, the main part of the ordinary (or similarly called everyday) business of the criminal network was picked out. Although it was likely that this part was not of particular interest for the police stakeholders (since it would just mock the already completed regular police investigations on the case), the rationale behind this decision was twofold:

- On the one hand, this aspect of the criminal network was simple enough for an initial modelling attempt in order to fully understand the involved methods, techniques and tools.
- On the other hand, it was anticipated that there existed strong dependencies between the ordinary business and the topic of the internal breakdown of the gang, so that this former aspect could reasonably complement the model of the latter — more important — topic[51].

In a quick sweep a rough shape of the money laundering process as part of the ordinary business was sketched (Figure 5.2). Firstly, as a precondition illegal money must be available, attained by any kind of criminal activities by a Black Collar criminal — drug or human trafficking, heist and the like. This

[50] This dynamic perspective is what sociologists are familiar with, as highlighted in (Neumann and Lotzmann, 2016a, p. 282): "Dies ist ein methodischer Ansatz, um Situationsbedingungen mit Anschlusshandlungen in Beziehung zu setzen, die dann wiederum neue Situationsbedingungen schaffen. Damit wird auf einer methodischen Ebene ein makro-mikro-makro Erklärungsansatz abgesichert (Greve et al., 2009). [...] Die Entschlüsselung der Sequenzen von Bedingungen und Handlungen folgt dabei einem bereits von Weber beschriebenen mikroskopischen Analyseschema 'das Gegebene so weit in Bestandteile (zu) zerlegen, bis jeder von diesen in eine Regel der Erfahrung eingefügt, und also festgestellt werden kann, welcher Erfolg von jedem einzelnen von ihnen [...] nach einer Erfahrungsregel zu erwarten gewesen wäre' (Weber, 1968, p. 279)."

[51] In the end it was decided to abstain from modelling these dependencies, mainly for simplicity reasons.

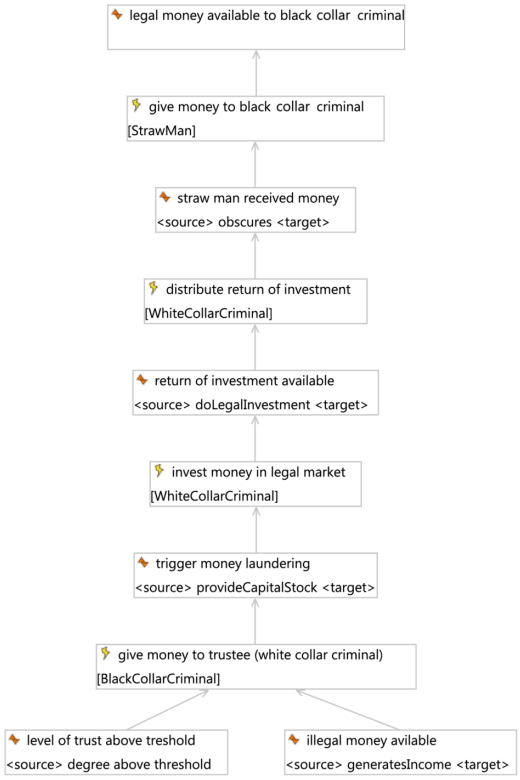

Figure 5.2: CCD Action Diagram of ordinary business

money is handed over to a White Collar criminal, a trustee with reputation both in the criminal and the legal world who would be able to accomplish investment activities to transform the black assets into legal money. The remainder of the process concentrates on this money laundering: With the provided capital stock the money laundering is triggered by the White Collar criminal by some kind of investment in the legal market, e.g. in construction projects or for operating amusement halls, both fields that seem to be prone for this kind of activities:

> "Finally, V01[52] paid 59 million. The cash money had been invested through a construction in Curacao. Here the brother of V01 played a decisive role." (Neumann and Lotzmann, 2016c, p. 163)

> "It is suspected that the amusement halls which are officially owned first by G[53] and then by V01 in reality are in the ownership of O1[54]."[55]

[52]V01 is the cover name of the White Collar criminal of the gang. See also section 5.5.2.
[53]G is the cover name of one of the known stooges.
[54]O1 is the cover name of one of the black collar criminals.
[55]CCD annotation for condition `legal money available to black colour criminal`.

After some (unspecified) time a return of investment can be expected from the legal investment, which then needs to be distributed among the Black Collar criminals. To obfuscate this kind of transactions, another actor comes into play: the stooge or — as called here — the 'straw man'. This is typically an inconspicuous individual with close relation to a Black Collar, e.g. a relative or an acquaintance, who performs the errand to hand over the money to the Black Collar criminal, so that the legal money is available for him.

Having this sketch, the question then was regarded if there might be any further conditions which would have to be fulfilled in order to maintain this everyday routine of the criminals. The first idea was that the criminal network needs to act covert, i.e. the existence of the criminal network is not known to representatives of the legal world (police, but also general public). This can be assumed as crucial factor for this particular kind of a criminal organisation in its particular social environment of a Western European country (where such activities are less endorsed than in other parts of the world), but there are many examples for criminal organisation where this factor is not preventing them to commit crime (e.g. Mafia-like organisations in Southern Europe, Asia or South America — see Anzola et al. (2016)). So this aspect was discarded (not modelled as a condition in Figure 5.2) for the sake of being as general as possible, an approach followed through the entire modelling enterprise (and which will become more evident later on). The second idea — which is already anticipated in the assignment of the title 'trustee' to the White Collar criminal — is the supremacy of trust. In more technical terms, a certain level of trust needs to be established in order to allow criminal business. Trust is mandatory among the actual members of the criminal network (i.e. the Black Collar criminals) and the supporting persons (like straw men), but in particular the trustworthiness of the White Collar criminal is crucial. If trust is no longer maintained, the stability of the network may be jeopardised. Here the actual matter of the model begins.

5.4.2 Starting Point: Crystallising kernel of mistrust

The inception of destabilisation within the criminal network is at the same time the starting point of the modelling activities where the qualitative analysis of the police files focussed on. It is assumed that a singular initial event can be sufficient in order to set off a chain of action that finally may result in a severe crisis of the criminal network. As the stability within the network crucially relies on trust among the members, the event is likely to have a diminishing effect on the trust among the criminals. Therefore, the starting point is modelled in a way that a member of the criminal network becomes disreputable, which means that some of the other criminals lose trust in this member. As the empirical evidence is not particularly clear about the actual

This and all following CCD annotations are anonymised citations from documents in the evidence base.

cause, two options are taken into account by the model: an unknown external event on the one hand, and a violation of an internal rule (or norm) the gang members have committed to on the other hand. A possible cause is mentioned in (Andrighetto et al., 2014, p. 48):

> "It makes the white-collar criminal disreputable if a story appears in the newspaper that financial transactions become public."

This is depicted in the starting `event: member X becomes disreputable` in Figure 5.3. As indicated in the condition box, the degree of a particular value assigned with trust (in the diagram expressed by the placeholder 'source') changes to low or even very low. Here a modelling decision based upon (Sabater-Mir et al., 2006) and confirmed in stakeholder discussions is reflected: Trust, on the one hand, can be assigned to an *image* value that one person has towards another person. The aggregation of the individual image assignments of a person are, on the other hand, expressed as *reputation*, a property known (or rather believed) by all members of the social system (here, the criminal network). From an operational point of view, image values are private knowledge by actors and have a dynamic character, i.e. these values are only stable in short-term periods and can change frequently due to any kind of experience. In contrast, reputation is considered as less dynamic (i.e. stable over long-term periods) global knowledge.

This design is reflected in Figure 5.3: The initial reaction on the event that a fellow criminal becomes disreputable is that the image of this member is changed to a lower level. According to evidence, typical reactions of fellow criminals towards a member with a tattered image are aggressive or even violent actions, coupled with the idea to sanction the deviating behaviour:

> "On Oct. 11, [...] at 10 am., the brother of O1, known as J.G. has been kidnapped [...]. The kidnapper stole the day's takings of two amusement halls and forced M and J.G. to sign IOU of 5 millions."[56]

> "The men are interested in murdering J.O."[57]

The result of such aggressions can basically be classified into two categories: Either the aggression has lethal consequences for the victim, or the victim survives the attack. In case of an assassination, a panic might strike the members of the criminal network, leading to cascading effects of aggression and violence. This case is subject of section 5.4.4. If the victim survives and becomes aware of the aggression (as drawn in the action diagram in Figure 5.3), an interpretation of this attack is required in order to find the reasons behind it:

[56]CCD annotation for action `perform aggressive actions against members X`; IOU stands for "I owe you".

[57]CCD annotation for action `perform aggressive actions against members X`.

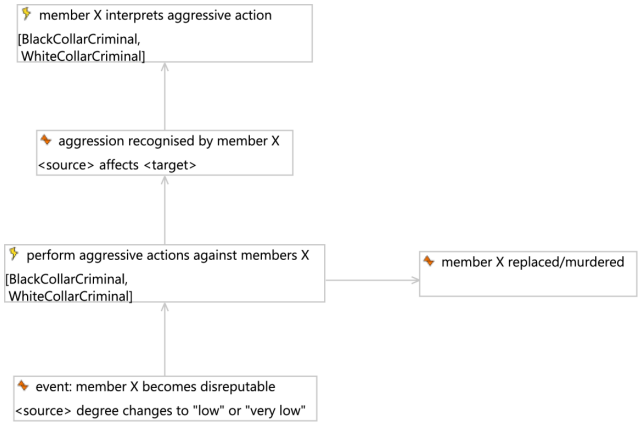

Figure 5.3: CCD Action Diagram of crystallising kernel of mistrust

> "If someone gets a machine gun pressed in his stomach and somebody is involved in this action [who] had been perceived as a friend, this may lead to an evaluation of the reason of the aggression." (Andrighetto et al., 2014, p. 48)

This interpretation process and the possible consequences are subject of the following subsection.

5.4.3 Escalation of internal conflict

The victim of a non-lethal aggression can be expected to respond in some way to the attack. The first phase of the response can be considered an interpretation process in which the possible causes for the attack are evaluated. In some cases the situation might seem clear during or immediately after the attack, when the person affected is aware of own misdemeanour, but this might be a misinterpretation. Often the causes remain blurred, in particular since aggressive behaviour among (emotionally acting) criminals is certainly a quite common thing. The abstract concept behind this interpretation can be formulated as a succinct question: Has the victim violated a social norm of the criminal network? Hence, a kind of normative process has to be considered in the model (Neumann and Lotzmann, 2016c).

This interpretation process is subsumed in the action `member X interprets aggressive action` in Figure 5.4, resulting in a decision that either the norm of trust is demanded (i.e. the confession that the victim has violated the norm of trust and that the aggression was a justified sanction), or that the norm of trust has been violated by the aggressor by doing an unjustified aggression. These two alternative decisions are modelled in the respective conditions `norm of trust demanded` and `norm of trust violated`, respectively. From these two conditions the range of actual response actions are predetermined.

In case of a demanded norm, the recipient of the sanction either obeys or cheats[58], expressed by the two actions `member X obeys` and `member X decides to cheat`:

> "I paid, but I'm alive."[59]

The case of a violated norm of trust sets off the 'more interesting' events in terms of the object of investigation: The victim in turn responds with an aggression. This aggression can either be a betrayal (as described below, Figure 5.5) or a counter-aggression. The action `member X performs counteraggression` reflects the latter and is based on the following phrases from evidence:

> "It seems plausible that a white-collar criminal has more resources to betray the organisation than to perform counter-aggression. He is the one who invested the illegal money and on which the investors have to rely. Thus he might easily cheat. However, when 'he was like a hunted cat' he also tried (unsuccessfully) to perform counter-aggression: 'He was at a point in which he was in a totally despaired situation. ... He had a plan to approach O1 with a weapon. However, in the last moment he didn't dare. At a different time he had two pistols with him. He planned to shoot O1 to death and to pass the other weapon in his hand (O1) in order that it appeared as if he had shot in self-defence.' Thus, he clearly identified who to attack and decided to (try to) murder him. However, aggression may remain on a lower level, such as kidnapping." (Andrighetto et al., 2014, p. 56)

The result is that a new `group member (new X) becomes victim of aggression` (which might be so serious that the `organisation becomes public` as well, e.g. in case of severe violence), as the respective conditions show in the model. Here a crucial detail in Figure 5.4 has to be pointed out: If the aggression was not lethal (i.e. the action `member X becomes disabled/murdered` and following condition `member X replaced/murdered` do not apply), the new victim of the aggression has, in turn, to interpret the aggressive action, possibly launching an escalation of aggressions and stoking up the internal conflict.

The alternative to perform a counter-aggression is to betray the criminal organisation — expressed in the action `member X decides to betray criminal organisation` (Figure 5.5). The related evidence is discussed in (Andrighetto et al., 2014, p. 54) as follows:

[58]Cheating in this context means that the sanctioned criminal pretends to obey, but in fact plots revenge. It has been included in the model as a hypothetical option for which indications, but no clear evidences could be found.

[59]CCD annotation for action `member X obeys`.

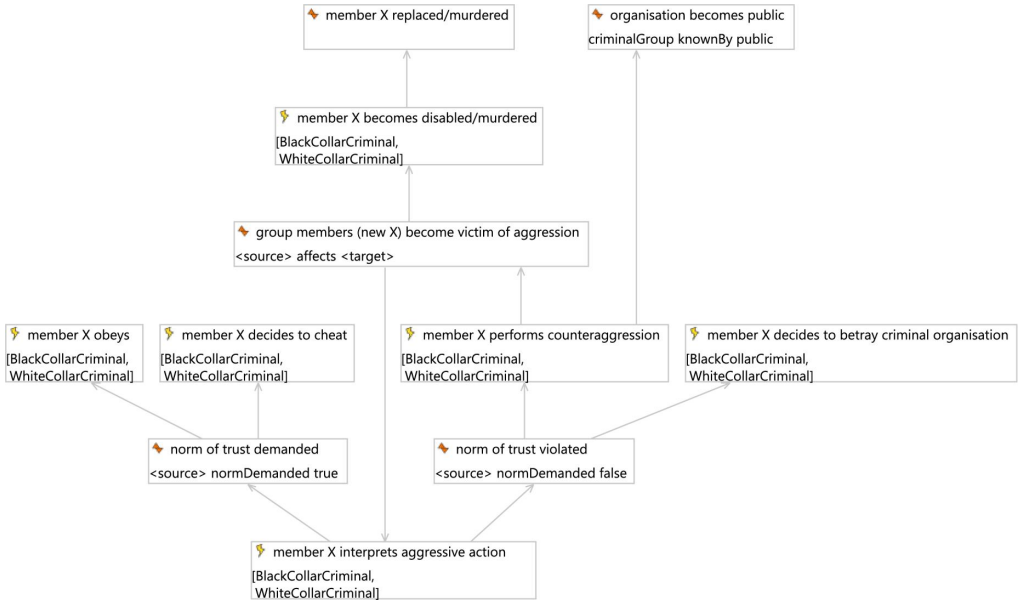

Figure 5.4: CCD Action Diagram of interpretation of and reaction on aggression

> "It is puzzling that the data seem to indicate that in the first instance betrayal had been undertaken by a black collar criminal (M): 'O1 wanted the money from her[60] and threatened M to kill him and his children. M told the newspapers [about my role in the criminal network][61] because he thought that I wanted to kill him to get the money.' Going to the newspapers is a severe violation of 'commitment to the norm of trust in the organisation', because this implies most notably to keep the organisation covert. It seems plausible that this was triggered by a certain element of panic, when he was threatened to be killed together with his children. The reaction entails two decision processes: first, not to keep the reaction inside the group and second, not to go to the police (which could have protected him), but doing something more nasty. On the other hand, some members went to police, eventually in order to try to save their lives or as a kind of heroic revenge, e.g., a few days before his murder V01 wrote a document with detailed information and gave it to his notary with the instruction: 'If I become liquidated I want the notary Mr X to hand over this document to the police. Signed at ZZ by V01.' "

[60] Presumably a relative of one of the killed network members.

[61] In the original quotation the role of V01 in the network had been revealed. For reason of protection of private data this role had to be anonymised.

Although in all cases of betrayal the trust of other members can be considered to be impaired (which is used as definition for the term betrayal here), these examples express quite different ways of betrayal. On the one hand there are actions which affect only individual criminals and their immediate surroundings, such as:

> "I belief it was in 1998 when V01 started an affair with N. (B19)"[62], the girlfriend of a another criminal.

Such 'nasty' acts are typical of small rebukes and may cause aggressive or violent reactions, but in the first place do not threat the stability of the network as a whole, as they are sustained within the covert criminal world. On the other hand, actions of betrayal can have severe consequences for the network if this line is crossed. 'Going to police' or 'Going to press' are two such examples, which can cause uncontrolled behaviour of the individual criminals, enacted by emotional reactions and panic:

> "V01, killed on May 17, [...] had spoken several times with criminal investigation officers before his death."[63]

> "... the affair with M. had been in the newspapers."[64]

Hence, the two categories of internal and external betrayal are distinguished in the model. The internal betrayal is abstracted as a single action `member X does 'something nasty'` with the possible result that the member X becomes disreputable. For the external betrayal, two actions are foreseen: one for the case that the `member X goes to public`, the other one for the case that `member X goes to police`. The former simply results in the criminal network (here referred to as organisation) becomes known to the public, the latter is expressed by a criminal complaint. The general public is not modelled as an actor, i.e. it is not regarded to take an active part in the simulation. Hence, the effect of the criminal network becoming public is that the police gets to know about the network. At the same time, all members of the network also become aware that the network is no longer covert, which might trigger certain actions. In contrast, a criminal complaint is a hidden act of one criminal actor of which the other members don't get a clue until some police actions strike the network.

The police is included as an (institutional) actor and becomes informed about the criminal network in both cases of external betrayal as well as indirectly by cases of severe violence:

> "At May XXX DLC had been liquidated at the exit G. of motorway AX."[65]

[62]CCD annotation for action `member X does 'something nasty'`.
[63]CCD annotation for action `member X goes to police`.
[64]CCD annotation for action `member X goes to public`.
[65]CCD annotation for condition `member X replaced/murdered`.

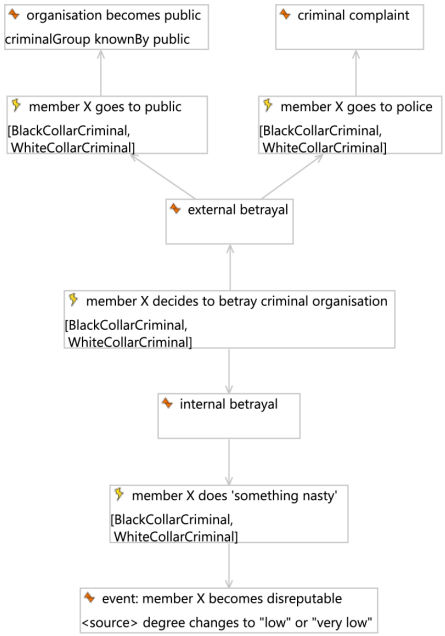

Figure 5.5: CCD Action Diagram of means to betray criminal network

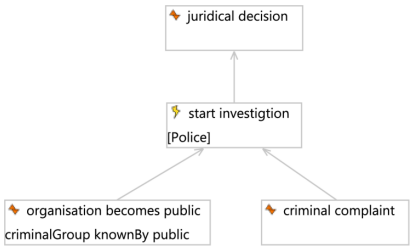

Figure 5.6: CCD Action Diagram of possible police intervention

As shown in Figure 5.6, the police `starts investigation` with the ultimate result of a juridical decision and the appropriate police actions.

5.4.4 A corrupt chaos

In the evidence base situations and sequences of actions are documented that by no means can be regarded as goal-driven, controlled behaviour of individuals — the members of the criminal network in this case. The occurred excessively violent and trust-undermining events are perceived by the involved persons as if "there is a corrupt chaos behind it"[66], which is also expressed in statements like "He was like a hunted cat"[67]. The loss of control of behaviour can be

[66] CCD annotation for action `panic reaction: fear for life (new X)`.
[67] CCD annotation for condition `fear for life`.

anticipated to happen due to exceptional emotional stress, basically generated by fear. In such a mental state of panic a human is focussed solely on existential needs, in particular to save the own life. The behaviour towards other individuals in the surrounding — even close friends — can become corrupted, and trust is no longer a controlling element. This results in a chaotic and threatening atmosphere:

> "'There is a rule of terror governing the town'. This means that the level of trust in the organisation is below a threshold in which the ordinary business could go on, i.e., a cascading effect of a spread of distrust generated an extremely bad reputation. This comes from the observation of betrayal and aggression in which at the end nobody was aware who initially violated the norm of commitment to the organisation and what aggression was intended as norm enforcement. Note that this might also include obedience as a reaction to aggression. At first sight this might appear as successful norm enforcement. However, it might decrease the subjective commitment to the group if one states: 'I paid but I'm alive'. To describe the situation in their own words: 'There is a corrupt chaos behind it'. This means that trust is not recoverable and actors are governed by panic." (Andrighetto et al., 2014, p. 59)

The conceptualisation of this complex facet of human behaviour is intentionally realised in an abstracting and very simplified way. It is based on the assumption that by focussing on observable events and reactions a 'corrupt chaos' can be reconstructed in the simulation, which in turn may allow to deduce the likely causes. This aspect of the model leaves room for experimentation with different cognitive heuristics, which could be tried out and evaluated in the formal model.

In the conceptual operationalisation as shown in Figure 5.7, a `panic reaction: fear for life` is triggered only in case of a murder of a fellow criminal. The resulting fear for the own life of member X than might trigger actions as exemplified above: `member X performs counteraggression`, `member X decides to betray criminal organisation`, and in addition also the possible (and according to the evidence very popular) case that `member X escapes`.

5.4.5 Run on the bank

While a state of panic due to fear for life may cause a corrupt chaos among the members of the criminal network, another kind of panic situation can be identified in the evidence: the fear to lose invested money. Although this can be considered as a less severe state of panic than fear for life, the resulting acts of aggression are quite similar in terms of severity:

> "The run on the bank is a result of the cascading effect of spreading distrust. If someone wants his money back and others get

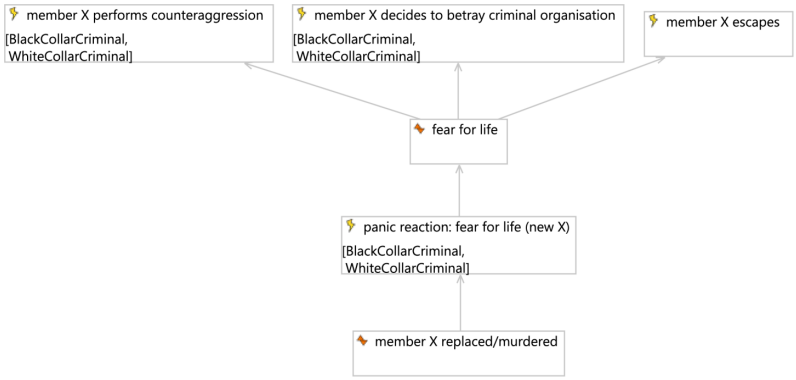

Figure 5.7: CCD Action Diagram showing the corrupt chaos

> notice of this, it is in their interest to get their money back as well. This threatens the liquidity of the bank, also in an illegal market: 'At a certain point he had problems with his liquidity.' This stimulated severe intimidation of the banker like getting a machine gun pressed in his stomach or commands such as: 'on May XXX, O5 came to my house in order to say that at 8 in the evening I should come to the forest. This is standard: intimidate and request for money.' " (Andrighetto et al., 2014, p. 61)

Actions linked to this situation appear in two different ways. On the one hand the aggressions are directed towards a specific target, the White Collar criminal in his role of the bank keeper. On the other hand, the focus of the aggressions is to get back the money, if need be by intimidating the White Collar. In the context of intimidation, often not only the own invested money is requested but it is also attempted to squeeze profit out of this opportunity by demanding additional amounts (which itself can fuel the spread of mistrust, as described above):

> "At the beginning of Oct. [...] S.K. came in the office. She told the employees that she needed to talk to me because her former man (who died) had 7 millions active debts. If I don't pay her [...] friend O6 would kill me."[68]

The reaction of the White Collar criminal is determining the subsequent course of events. If he pays the demanded money the situation can calm down again and even turn back to a 'normal' state. On the other hand, the situation can escalate in a 'run on the bank' if the money is not returned or the intimidation becomes known to other criminals.

Figure 5.8 shows how the conceptual model covers these aspects. The starting point is the `panic reaction: fear for money` on an arbitrary member

[68] CCD annotation for action `member X requests invested money back`.

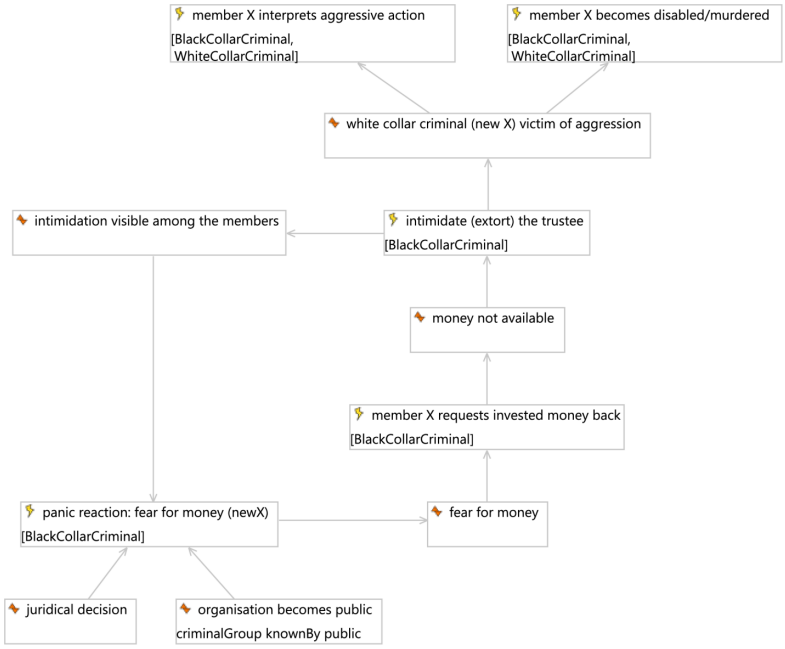

Figure 5.8: CCD Action Diagram of 'run on the bank'

X, caused by either a juridical decision (and the resulting act of detention), or by the fact that the organisation (i.e. criminal network) becomes public. This condition of fear for money then triggers the action that `member X requests invested money back`. In the case that the money is not available this is considered to be a violation of the norm of trust, and, hence, member X starts to `intimidate (extort) the trustee`. This intimidation is performed by an aggressive or violent action against the White Collar criminal, who thereby becomes victim of aggression. This results in consequences as already described above: Either member X (who is the White Collar criminal in this case) does not survive the attack (action `member X becomes disabled/murdered`), or the cause of the attack needs to be interpreted (action `member X interprets aggressive action`), with the possibility of a counter-attack by aggressive actions. The refusal to pay back money by the White Collar might also become known to other members of the network which, in turn, might trigger a panic reaction on their side. This cycle constitutes the basis for the cascading effect of a 'run on the bank', i.e. an increasing number of criminals try to get hold of money from their trustee.

5.4.6 Evaluating the state of trust

Finally, a process relevant at all stages of the internal conflict within the criminal network is included: the evaluation of the state of trust within the network. As initially stated, trust is the prerequisite for successful ordinary criminal

business, undermining of trust makes the network prone to conflict.

Figure 5.9 shows the related action `evaluate level of trust within criminal organisation` with the preconditions and the possible consequences depending on the evaluation outcome. The evaluation is performed (i.e. the action is triggered), if one of the following conditions is given:

- One of the members of the network has been murdered.
- The norm of trust has been violated, as a possible result of the interpretation of an aggressive action (see Figure 5.4).
- In contrast to the previous two conditions which are likely to cause decline of trust, there is also a condition where the level of trust might increase: a member of the criminal network obeys a sanction (action `member X obeys`).

The evaluation can either result in the belief that the level of trust is above the threshold (which is subjective to the individual), or that the threshold is not reached anymore. In the former case the evaluating criminal will continue the business undisturbed, while in the latter case the attention will shift towards the cause of the loss of trust. This can happen in different ways.

A rather rational way to solve the problem and re-establish trust would be to find the responsible individual X (action `find responsible member (new X)`) so that the loss of trust can be encapsulated around X: This member becomes disreputable, which goes along with a decrease of image (triggering sanction measures according to 5.4.2). The option is a panic reaction (action `panic reaction: unspecific (new X)`[69], a situation where emotions rule out rational behaviour and the criminals get into a fear for money or for life.

This process concludes the conceptualisation of model dynamics found relevant by discussing the results of the qualitative analysis of police documents between Data Analyst, Simulation Modeller and police stakeholders. The following section complements this perspective with other elements crucial for the envisaged transformation into a simulation model.

5.5 Steps towards model formalisation

The action diagram sketched step by step in the previous section is the result of intense collaboration between Data Analyst and Simulation Modeller, amended by input from stakeholders emerging from discussions about intermediate results. The model represents the state of 'theoretical saturation', i.e. it is assumed that the aspects relevant for the subject of the model are covered and reflect the evidence data in an appropriate way. Hence, this part of the conceptual model has fulfilled its purpose as a communication tool to discuss

[69] The 'new X' in this action refers to possible spread of this information, where another member of the network panics e.g. due to rumours.

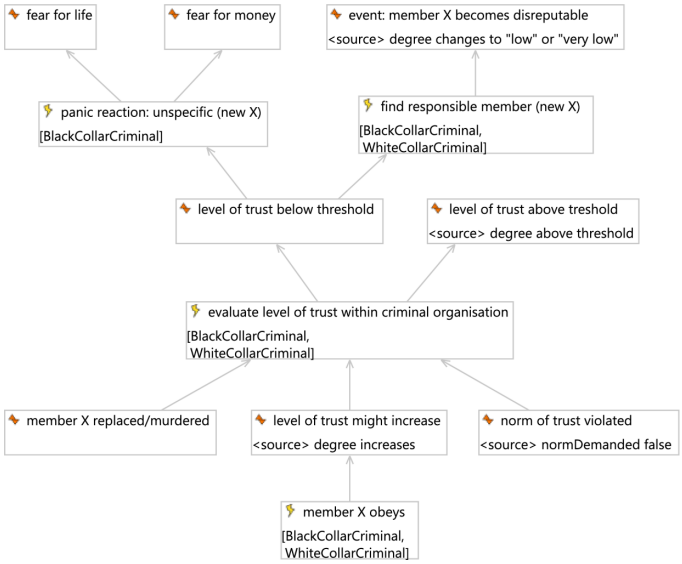

Figure 5.9: CCD Action Diagram for evaluating the state of trust, together with triggering events and consequences

the case in a structured way and to shape the modelling steps ahead. Now the phase of constructing the parts of the conceptual model necessary for further formalisation can be tackled.

The following subsections show these formalisation aspects in order to provide an understanding on the degree of formalisation and the level of detail that can be handled, given the preconditions and constraints set by the CCD modelling approach. It has to be noted that the picture drawn here should by no means be understood as an objective gauge, but as strongly dependent on the particular modelling subject and even more on the personal capabilities and preferences of the person conducting the modelling (in this case the Simulation Modeller). In other words, an example is given what a practicable approach for finding a cutting line between conceptual modelling and model programming can look like.

5.5.1 Model structure

The model dynamics presented in the previous section require a suitable backing from the static perspective, meaning that appropriate representations of physical and mental entities need to be modelled as well. This process of creating a reasonable model structure can be done as a 'supporting activity', i.e. providing the necessary entities in order to model the intended dynamic elements. In this concrete case of collaborative modelling, these static aspects were put together by the Simulation Modeller in form of an actor-network-diagram as part of the CCD. This diagram type basically contains

the elements populating the simulation world, actors and passive objects with their attributes, as well as relations between these entities. During the first iteration of discussing and modelling the dynamics the creation of the actor-network-diagram was done as a 'background task' without deeper discussion with the DATA ANALYST. As a result, an initial action diagram sketching a coarse shape of the involved intra- and inter-agent processes together with a first version of the actor-network-diagram were available.

The actor-network diagram for which a walk-through is presented in the following paragraphs is, however, the final version used to implement the model. Hence, not only the supporting scaffold for the action diagram is covered, but also additional entities are included to support the model implementation. The diagram is not presented here as a whole, but the most important core concepts and design decisions are described. The full set of diagrams can be found in the source code of the anonymised simulation model (see annex B).

The Criminal Network

The *criminal network*[70] as the encompassing concept for this model is designed as a CCD object. In Figure 5.10 it is represented by the box in the centre, marked with the symbol ▫ (the identification for objects in the CCD actor network diagram; for further information on the CCD see section 3.3.2). Two actors are directly related to the criminal network: The *criminal* and the *police*. Both actors refer just to the concepts of this actor-type, without any hint about the e.g. multiplicity (which is modelled by instance definitions, see also 5.5.2). Actors are marked with the symbol ●. The single relation between the criminal and the network covers the trivial case that a criminal is a member of the criminal network. Between police and the network three relations are shown, telling on the one hand that the police knows (or rather is able to know) the criminal network, but on the other hand (the perspective of the criminal network) that the network might be known by the police, and, furthermore, the possibility that the criminal network might try to undermine the police activities by corruption.

There is another entity which could also be regarded as an actor, but is modelled here as an object: the *public*. Rationale for this decision is that the general public is not expected to play any kind of active role in the model, but rather serves as a passive concept with just the functionality as a bearer of information about the existence of the criminal network. I.e., the public might know about the criminal network (and 'leak' this information to the police), and the criminal network might be aware to be known to the public (so, again the two perspectives of this identical information).

A completely different type of object related to the criminal network is the

[70] The slanted text formatting for some phrases in this and the following paragraphs is used to refer to concrete CCD elements that are part of the actor-network-diagram, from which fragments are used as illustrations in this section.

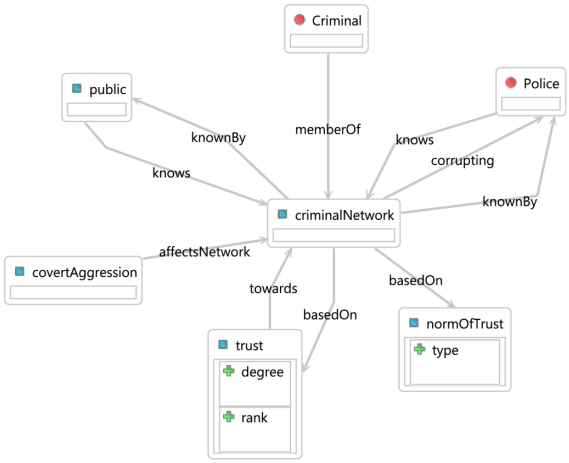

Figure 5.10: CCD actor-network-diagram showing the intertwining of the criminal network

covert aggression. This is a type of aggression performed by criminals, invisible to the public, but affecting the internal state of the criminal network. There is also a possibility that information about this kind of aggression is leaked to the police — this will become clearer in the course of this section.

There are two more objects shown in the diagram fragment on which the criminal network is based upon: *trust* and the related concept of a *norm of trust*. The norm of trust here represents the concept of a social norm, defining that the members of the criminal network have to trust each other in order to maintain stability, where different types of norms exist (depicted as an attribute of this object, the element with the ✚ symbol). This normative approach is subject of the subsection 5.5.4 below. On the other hand, the concept of trust is a more technical realisation of this norm: it expresses the trust toward different actors and concepts, as described in the paragraph on Concept of Trust below.

The Criminal Actor

For the main actor of the model — the *criminal* — the directly associated entities are depicted in Figure 5.11. Apart from the role as part of the criminal network (as defined in the previous paragraph and not shown in the diagram), various related entities are included: sub-actors, properties and activities. These will be described subsequently, beginning with the sub-actors (top-left) and onwards counter-clockwise in the diagram.

First of all (and as informally introduced before), the concept of a criminal considered here can be further distinguished into *Black Collar* and *White Collar* criminal actor types. The 'is-a' relation reveals this sub-concept rela-

tionship. The White Collar criminal has a primitive[71] attribute for a *capital stock*, the amount of money available to be paid back in the final stage of the money laundering process (this is a hint for the technical realisation of the run-on-the-bank incident).

Next, a couple of objects are specified for marking the criminal's condition:

- The physical *state* of the actor with the values *active*, *arrested* and *dead*. One of the three values can be assigned to the *state* attribute, defined in a CCD enum, not visible in the diagram. The respective object is called *agentState*[72] to emphasise the technical motivation for this element.

- The *mental frame*, where the criminal can either be in one of the states defined by the sub-objects *rational* or *emotional*.

The next (again technically motivated) object describes the result of the cognitive process of *norm evaluation*, where after a norm evaluation it is stated if the *norm of trust* is demanded in this particular situation or not. The dashed red arrow 'known-by' points out that this object is only visible to the criminal who does this norm evaluation. The known-by has to be understood as a property of instances, not concepts: Each instance of an object can only be known by a particular instance of an actor.

The following two objects *reputation* and *image* are norm-related properties of a criminal, considered as important parameters for various decisions modelled in the action diagram. These concepts are defined and motivated in section 5.4.2 and also operationalised in 5.5.4. The reputation is a property of each criminal, i.e. each of the individuals (instances of the concept criminal) has a reputation of a certain degree. For this, an ordinal-scaled attribute is defined with the values *very low*, *low*, *modest*, *high* and *very high*. Reputation is considered as long-term stable[73] and well-known to other criminals, at least among the members of the network. Included in the diagram is again the object *trust* since (among other factors) it emerges from reputation. Trust — from the perspective of the criminal — is an individual belief, and as such modelled as private knowledge by the known-by relation. This construct reflects the individual trust assessment of a criminal. This concept in general is discussed a bit further in the next paragraph. Image is also a property of each criminal of a similar flavour and with comparable impact as reputation (hence with the same kind of degree attribute), but it is considered to be less stable and subject to change in the course time, with relation to interactions

[71] Primitive attribute are just values, in contrast to complex attributes that can be constructed via further objects attached by relations

[72] 'Agent state' refers to the state of the agent object in the simulation model, instead of the from a CCD perspective more logical 'actor state'.

[73] Long-term stable can be interpreted as 'beyond the time scale of a simulation run' (or even of real life). In the actual model implementation this value is kept constant, mainly for reasons to reduce the number of variables in the model.

among the criminals. Furthermore, the image is a private property, reflecting that each criminal has an individual view on the image of each other criminal.

Moving on, the *aggressive action* is one of the core concepts and plays a major role in the dynamics specified in the action diagram. It encapsulates behavioural and communicative aspects of criminals. As the diagram tells, the aggressive action is performed by a criminal (individual X), and at the same time is directed to another criminal (individual Y). Aggressive actions are categorised according to their strength and frequency. Strength is a measure of the potential impact (and hence the subjective severity) and can be either *low*, *modest* or *high*; frequency is a measure derived from the empirical number of appearances of the different kinds of aggressive actions. Hence, this concept is strongly related to results from the qualitative text analysis. A paragraph of section 5.5.2 (p. 133–135) is dedicated to the parameterisation of this concept, i.e. the considered instances with strength and frequency values.

Two further objects describe events which are modelled to be not actively performed by a criminal, but nonetheless have important impact. Firstly, there is a *normative event*, something that can be associated with behaviour of the criminal. For example, certain actions that are considered to breach the norm of trust will trigger related normative events. This object is technically motivated (like other types of events, see below) to have a concrete data element to be memorised in a normative information repository. In contrast, the object *crime* is empirically motivated. This is a container for activities of the everyday business of the criminals, like drug trafficking or money laundering. The related instances of crime are derived from the evidence base, as shown in 5.5.2 (p. 135–136). This particular concept is assumed to be performed by criminals and to involve (or affect) criminals, but intentionally no relation directed from Criminal to crime is present, as details of the criminal business are not incorporated in the model dynamics.

Concept of Trust

Figure 5.12 provides a deeper look into the concept of trust already mentioned in the previous two paragraphs. *Trust* in a fellow criminal can emerge from *reputation*. Hence, a highly reputable criminal might be one who can be trusted. But trust is considered here as a more general concept: Not only trust towards another *criminal* is regarded, but also trust towards the *criminal network* as the representation of the social environment, towards the other actors *straw man* and *police*, as well as towards the *public*. Trust as defined in the actor-network-diagram is a technical means to capture an actual *degree* in order to construct a ranking of trusted entities. For the attribute *degree* an ordinal scale applies, with the following values:

- The value *established* denotes the optimal state of trust.
- With *recoverable* a state is marked where trust is impaired, but not

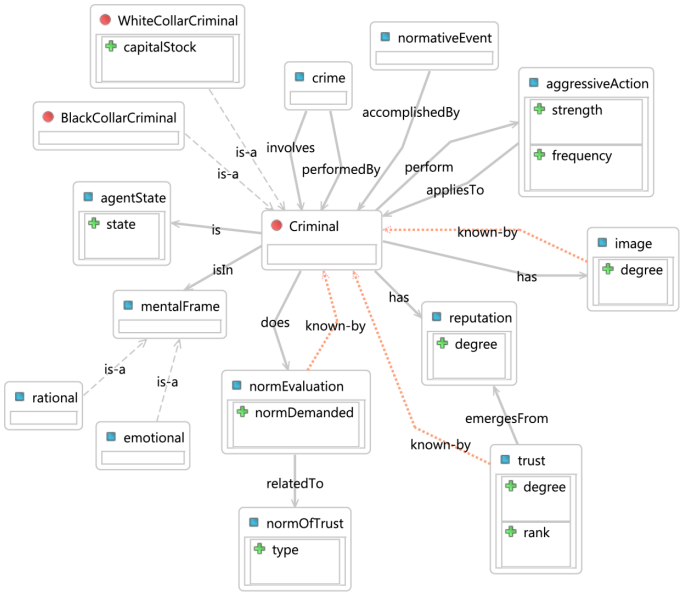

Figure 5.11: CCD actor-network-diagram showing the attributes and activities of a criminal

to a degree that would make re-establishing of a trusted relationship impossible.

- Trust is considered *not recoverable*, if a relationship is severely damaged and cannot be recovered.

The *rank* attribute can adopt an arbitrary integer value, so that a hierarchy of trusted entities can be constituted. The calculation of this value is based on the trust towards the entities and the time interval of changes of trust values. For example, if trust in in the *criminal network* and the *police* have the same value (e.g. *established*), but trust in *police* more recently changed from *recoverable* to *established*, then *police* has a higher rank.

This concept of changing trust over time towards the different parties is included in the conceptual model in order to support an approach for an emotion-driven evaluation of the current state the trust. The result of this evaluation influences the decision to betray the criminal network[74], i.e. to turn towards a more trusted party, for example the police or the public.

[74] Although foreseen in the CCD, this concept is currently not used in the implementation, but a possible starting point for a model extension. This is an example where during the implementation it was discovered that this mechanism has a quite low relevance to the object of investigation. The model subject was refocussed towards the role and emergence of image in view of the sanction recognition process.

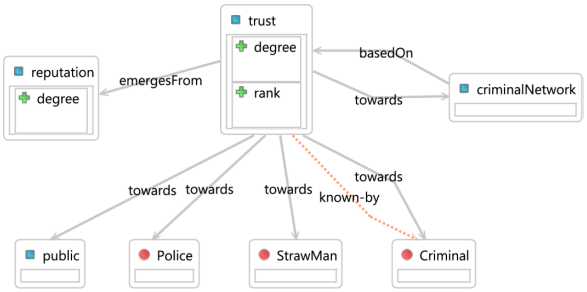

Figure 5.12: CCD actor-network-diagram showing the concept of trust

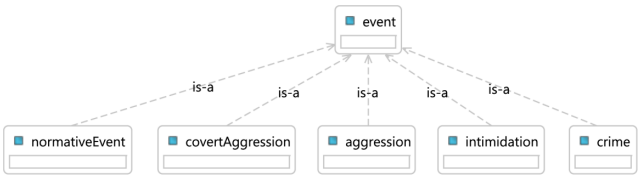

Figure 5.13: CCD actor-network-diagram showing events related to activities of criminals

Events

Figure 5.13 gives an overview of the types of events considered in the model. An *event* is signalling direct or indirect results of a criminal's activity. Some examples have already been mentioned above: *crime*, *covert aggression* and *normative event*; additionally there are *aggression* and *intimidation*. Crime is a placeholder for everyday activities of a criminal. Aggression and covert aggression are interlocked with aggressive behaviour among criminals, similar to intimidation which is a special case of aggression against the White Collar criminal in the situation of forcing to surrender money. In contrast, a normative event is an indirect consequence of another action or decision of a criminal, e.g. a norm violation associated with an unjust aggression. All these events are detailed in the following four paragraphs.

The concept of the event has been introduced in order to provide an overarching means to make the actual accomplishment of various activities tangible, i.e. to formalise simulation states in order to enable actors to evaluate consequences. Hence, it is a technical construct associated to conceptual entities predominantly derived from evidence, as the subsequent paragraphs will illustrate.

Crimes

Crime is an event depicting the ordinary business of criminals. According to Figure 5.14, a crime is accomplished by means of a *criminal action*, performed

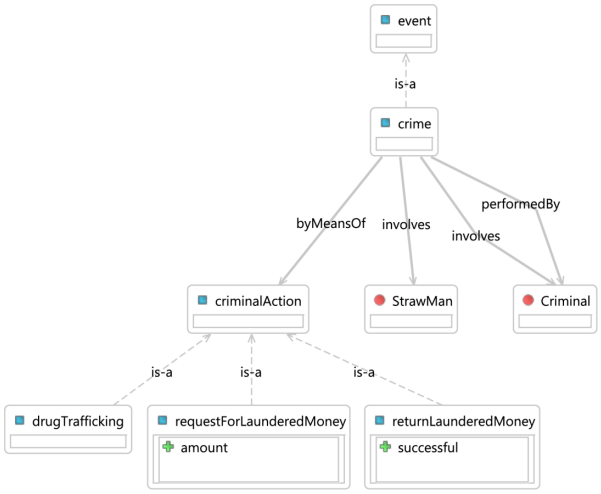

Figure 5.14: CCD actor-network-diagram showing the crimes performed by criminals in the everyday business

by *criminal*s (perpetrator perspective), involving other *criminal*s (perspectives like accomplice, victim etc.) and *straw men* (covering the criminal action).

The criminal actions considered in the model are all related to financial affairs of the criminals. On the one hand, the matter of generation of income is reflected in the action *drug trafficking*. As said before, this topic is not in the scope of the model — just an abstract source of black money is needed. The process of transforming the black money into legal money is, in contrast, one of the core subjects of the model. For this purpose, two actions are foreseen: A *request for laundered money*, attributed with an amount of requested money, and the *return of laundered money*, with an attribute that records the success of the action. These two actions are typically parts of interactions between Black Collar and White Collar criminals. A detailed picture of the concept of money laundering is drawn in the paragraph Aspects of Money Laundering.

Aggressions

The events related to aggression and the associated entities are sketched in Figure 5.15.

One event is the ordinary *aggression*, a typical mode of interaction between criminals, affecting identifiable criminals by causing harm on (at least) one side. As formulated in the Action Diagram (see section 5.4), reasons for an aggression can be issuing a sanction, some kind of self-interest (or even an unmotivated arbitrary incident), or a reaction on a sanction or an arbitrary aggression.

Means of an aggression is the *aggressive action*, with certain *strength* and *frequency*, performed by and applied to a *criminal* (these attributes and rela-

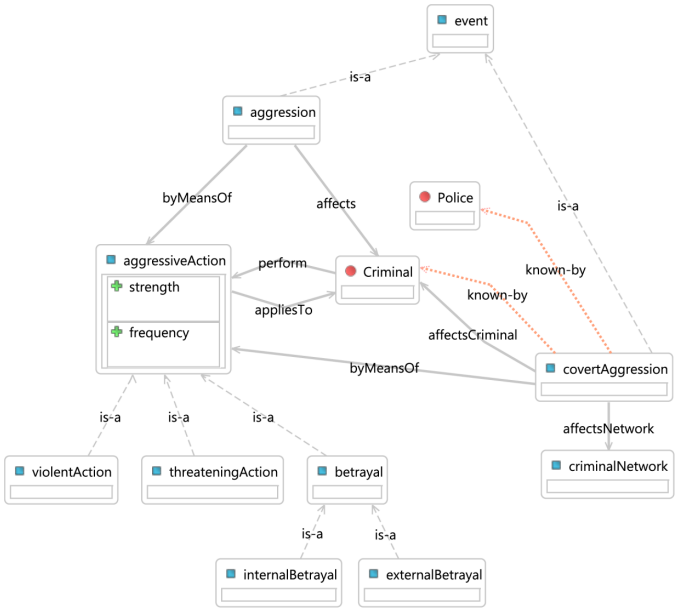

Figure 5.15: CCD actor-network-diagram showing the aggressions in which criminals can be involved

tions are described in the paragraph about the The Criminal Actor above) . The categories of aggressive actions conceptualised in the model as sub-objects of aggressive action are reinforced by the evidence base and reflected in the diagram: *violent* and *threatening* actions as well as *betrayal*, appearing in the two shades of *internal* and *external* betrayal. Examples in the paragraph on Aggressive Actions will motivate and illustrate the selection of these categories.

A special case is the event of a *covert aggression*. This is also performed by means of an aggressive action, can also affect individual (identifiable) criminals. The affected criminals can either be a rather small subset of the network members, but there are also scenarios where the whole criminal network is affected without the possibility to identify the criminal responsible for the aggression. An example for the former case is the internal betrayal where just two members of the network are in clash (and know about the aggression; this is expressed by the known-by relation to *Criminal*). Going to police is an example for the latter case, where the known-by relations to *Police* and *Criminal* come into play, i.e. only the criminal making the complaint and the police receiving the complaint know about the aggression against the criminal network.

Normative events

As shown in Figure 5.16, the *normative event* is accomplished by a *criminal*. The kind of the normative event is defined by a *normative action*, another

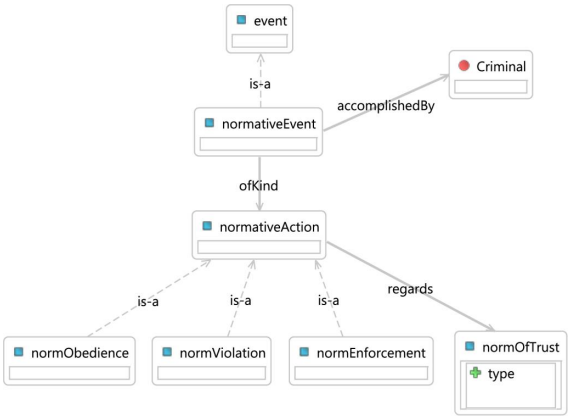

Figure 5.16: CCD actor-network-diagram showing the considered normative events

object which establishes the connection to the *norm of trust* object. Three different kinds of normative actions are included in the conceptual model, which can be classified in two groups: On the one hand there are the two indirect normative actions *norm violation* and *norm enforcement*; indirect in a sense that such actions are coupled to other actions performed by a criminal that can be perceived (or interpreted) as enforcing or violating a norm. An example is the aggressive action that can be either a justified sanction or an arbitrary assault, depending on the actual situation. On the other hand, the *norm obedience* is a direct normative action, where a criminal acknowledges an aggression as a sanction and respects the norm of trust.

The normative event is the central concept for the operationalisation of decisions based on norm conforming or norm deviating behaviour in the simulation model. Similar to the aggression it can be memorised in a normative information repository and evaluated (i.e. set into relation with an aggression) in the normative reasoning. This interplay of the different types of events is addressed in 5.5.4.

Aspects of Money Laundering

The last type of event — the *intimidation* — is presented in context of the only situation where it plays a role: the business of money laundering. Figure 5.17 gives an overview on the involved concepts. *Drug trafficking* generates income to the *Black Collar* criminal, which has to be turned into legal money, as described before in section 5.3. The action diagram presented there in Figure 5.2 presents money laundering as a 'black-box action', which can be imagined to rely on the following assumptions and process: The Black Collar criminal trusts in a *commitment* guaranteed by the *White Collar* criminal, the actor responsible for money laundering. This commitment is based on the *norm*

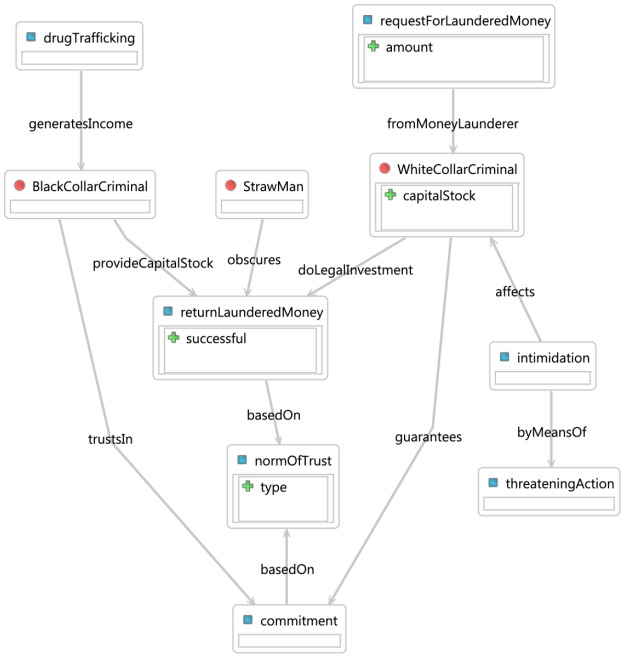

Figure 5.17: CCD actor-network-diagram showing the concepts relevant to money laundering

of trust, as is the act of *returning the laundered money back* by the White Collar to the Black Collar criminal. The latter is the important aspect to be operationalised in the simulation model. For this act a respective object is defined in the Actor-Network-Diagram in Figure 5.17. In this abstraction only the prerequisites for a successful return of money are reflected: the capital stock provided by Black Collar criminals, the investment of the capital in the legal market by the White Collar Criminal, and the activities to obscure these activities performed by *straw men*.

The more relevant case (for the model) of requesting money back from the White Collar criminal is expressed by an object for a *request for (a certain amount of) laundered money*. Although not shown in this diagram (but which can be deduced from Figure 5.14), this object is a kind of a *criminal action*, i.e. performed by a criminal typically of the Black Collar kind. Here the *intimidation* comes into play, if the payback does not run smoothly. In this case a *threatening action* against the White Collar criminal might be applied.

The Police actor

The other important actor type regarded in the model is the *police*. In contrast to the criminal actor, the design only covers aspects where the relation to the criminal network or the individual criminals is relevant and ignores details on internal processes. According to Figure 5.18, main task of the police is to

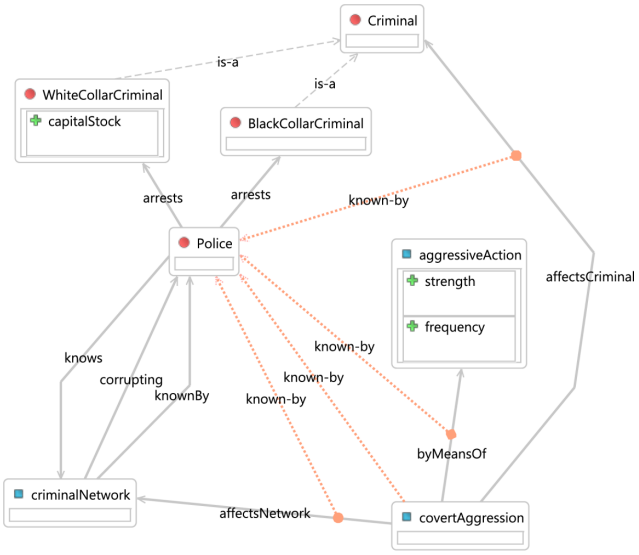

Figure 5.18: CCD actor-network-diagram showing the embedding of the police in the conceptual model

arrest criminals of both kinds (i.e. *White Collar* and *Black Collar* criminals). In order to take such effective measures, knowledge about the *criminal network* is needed. This knowledge can be gained in different ways: Besides publicly available information (external betrayal 'going to public' or the results of violence), the object *covert aggression* plays a central role in this regard. This is information about *aggressive actions*, about *criminals* or the *criminal network* as a whole, passed over to the police by a member of the criminal network, e.g. by making a criminal complaint. The known-by relations displayed in the diagram highlight this covert nature of this knowledge, i.e. the other members of the network are not aware of this situation.

There is one other relation between criminal network and police included: an abstract means of corrupting the police by the criminal network. This potential risk is not developed further even on the conceptual level as the importance appeared too marginal, but has some backing in the evidence base:

> "The source of the information leakage cannot be located. Neither it is known where the criminal got the information that he could tell 'that there are leakages in the police.' It is known, however, that 'O1 had the lawyer totally under his control' and a lawyer has access to court files." (Andrighetto et al., 2014, p. 63)

5.5.2 Parameters in the Conceptual Model

The conceptual model discussed so far covers structural and dynamic aspects only on a class level. This means that just general concepts of actors (like

criminal), actions (like aggression) and so on have been defined, not the actual instances of these concepts like e.g. the concrete number and 'character' of criminals, or concrete aggressions like kidnapping or attempted murder. This kind of information can be added on the conceptual level, too, and can be seen at the same time as a means of parameterising the model.

This conceptual parameterisation is important for the design of the model, as decisions like how the actors will be represented by agents in the simulation model (e.g. microscopic or institutional) are made here. This step is also crucial with respect to the evidence-driven modelling approach (and traceability), since indications for these concrete instances more or less directly reflect the results of the qualitative analysis process, and hence the facts from the evidence base. To illustrate this part of modelling, a default instance configuration will be described in this subsection, i.e. the set-up used for simulation model testing and a subset of the productive simulation runs. The most important concepts for which instances have to be defined are introduced below.

Actor instances

First, the criminal network is represented by an instance with the name 'criminalGroup' of the object criminalNetwork (i.e. the existence of exactly one network is considered here). The criminal network is modelled as a 'passive institution', hence an object is used instead of an actor. The individuals participating in the network are ten Black Collar criminals and one White Collar criminal who are mentioned by (cover) names in the evidence documents, but received neutral names in the model:

- 'ReputableCriminal' and 'Criminal', both of actor type Black Collar criminal:

  > Cover names for these criminals in the evidence documents are (among others): O1, O5, O6, O8, O9, B20, B30, SK, JM and DJL[75].

- 'WhiteCollar' of actor type White Collar criminal:

  > The cover name of the White Collar criminal is V01[76].

ReputableCriminal and Criminal are two different instances of the Black Collar criminal actor with different counts and characteristic initial attribute settings. The ReputableCriminal represents an (at least initially) more rational and less aggressive nature compared to the Criminal, assuming that decisions and commands of these are more often accepted and followed by network members as an effect of the higher reputation (see subsection 5.4.2). The criminal actor with the highest assumed reputation is the WhiteCollar

[75] CCD annotations for actor `BlackCollarCriminal`.
[76] CCD annotation for actor `WhiteCollarCriminal`.

Parameters	**ReputableCriminal**	**Criminal**	**WhiteCollar**
Count	7	3	1
Name	'ReputableCriminal-1' to 'ReputableCriminal-7'	'Criminal-1' to 'Criminal-3'	'WhiteCollar'
Mental frame	rationalFrame	emotionalFrame	rationalFrame
Reputation[77]	high	modest	very high
Membership	criminalGroup	criminalGroup	criminalGroup
Remarks	—	—	attribute 'capitalStock' is initialised with a value of 20 million (unspecified monetary units)

Table 5.1: Parameter settings of criminal actor instances

criminal, for whom (according to the evidence) just one instance is foreseen. The attributes for these criminal actor instances are abstractions based on cognitive heuristics; detailed attribute settings are collected in Table 5.1.

For the Straw Man no instances are defined as this actor does not play a significant role in the internal conflict of the criminal group (or at least there are no sufficient indications in the evidence which would justify the additional modelling efforts). So the remaining prominent actor is the Police. Since only a very limited and abstract set of interactions between the criminals and the police is modelled (see 5.5.1), a realisation as an institutional agent is feasible. Hence, just one instance 'Police' is defined for this actor. Further parameters are not regarded at the conceptual level.

Similar to the criminal network, the public is regarded as a passive institution, represented also as an object for which exactly one instance 'public' is defined.

Norm instances

The norms are the abstract foundation for concepts (like trust), for more tangible actions (like norm obedience, norm violation and norm enforcement), and in terms of cognitive processes (like norm evaluation) the basis for decisions. The concept related to norm is the object normOfTrust, for which instances are defined reflecting the following hierarchy of norms:

1. Moral norm 'Commitment to norm of trust in organisation'. This norm rules all criminal and straw man actors and can be formally described as

[77] Modelled via another object 'reputation', for which the initial reputation values of the different actors are specified as object instances.

$$NORM(1) \mapsto NOT\ VIOLATE\ TRUST\ c\ o \tag{5.1}$$

where c stands for criminal and o for organisation. This is a top level norm for which four concrete obligations are formulated:

(a) Obligation 'Distribute return of investment' (within process of money laundering). This norm rules the White Collar criminal actor and can be formally described as

$$NORM(1.1) \mapsto DISTRIBUTE\ c^w\ m\ s \tag{5.2}$$

where c^w stands for White Collar criminal, m for money and s for straw man. This norm describes the obligation that the White Collar criminal has to hand over the return from money laundering activities to a straw man in order to covertly transfer the money to the Black Collar criminals.

(b) Obligation 'Give money back to black collar criminal' (within process of money laundering). This norm rules the Straw Man actor and can be formally described as

$$NORM(1.2) \mapsto DISTRIBUTE\ s\ m\ c^b \tag{5.3}$$

where s stands for straw man, m for money and c^b for Black Collar criminal. This norm describes the obligation that the straw man has to hand over the money received from the White Collar criminal to the Black Collar criminal (and not keep it to himself).

(c) Obligation 'Punish deviant members'. This norm rules all criminal actors and can be formally described as

$$NORM(1.3) \mapsto PUNISH\ c_i\ c_j\ IF\ c_j\ VIOLATE\ NORM(1) \tag{5.4}$$

where c stands for criminal. This norm describes the obligation that criminal i has to punish criminal j, if the latter violated the moral norm 'Commitment to norm of trust in organisation'.

(d) Obligation 'Keep organisation secret'. This norm rules all criminal and straw man actors and can be formally described as

$$\begin{aligned} NORM(1.4) \mapsto\ & NOT\ DENOUNCE\ c\ o \\ & \wedge NOT\ PERFORM\_EXCESSIVE\_VIOLENCE\ c_i\ c_j \\ & \wedge COVERT\_MONEYLAUNDERING\ c^w\ s \end{aligned} \tag{5.5}$$

where c stands for criminal, o for criminal organisation, c^w for White Collar criminal and s for straw man. This norm describes the obligation that neither the criminal network shall be denounced to the

police or public, nor that excessive violence shall be performed between criminals (that might be publically recognisable), nor that money laundering activities which involve White Collar criminals and straw men shall be uncovered.

2. Prohibition 'Not pass information to criminals'. This norm rules the police actor and can be formally described as

$$NORM(2) \mapsto NOT\ PASS\_INFORMATION\ p\ c \tag{5.6}$$

where p stands for police and c for criminal.

These norms are a minimal set regulating the interactions between criminals as well as criminals and police in the environment of the particular criminal network modelled in this use case with the modelling aim to only cover the internal conflicts. The formal definition above is aligned with the approach followed in GLODERS for other use case models as defined in (Andrighetto et al., 2014). It is chosen to be compatible with the related concept of a normative process as sketched in 5.5.4 and to be prepared for a mathematical formalisation on the normative decision processes as applied in (Nardin et al., 2016a)[78]. There, also the actually intended realisation in the simulation model is described.

To migrate the model into other environments, i.e. to model interactions with the society or to focus on criminal organisations embedded in different cultural systems, the starting point would be to define the relevant set of norms, in order to define the related actions (subject of subsequent paragraph) and to ultimately shape the implementation of the decision processes (introduced in section 5.5.3).

Action instances

The largest group of object instances taken from evidence are related to various kinds of actions in which the actors are involved. The selection of these instances has large influence on the 'recognisability' of simulation results by stakeholders, but are less important in terms of the result quality. This means that the simulation model could also be verified with instances describing abstract actions of different categories and spectra of attributes (e.g. severity and frequency/probability), but for model validation the resort to real actions of a field the stakeholders are familiar with can greatly improve this step. Furthermore, readability and linking simulation results to evidence documents via traceability benefits from using real-world actions.

The following paragraphs give an overview on the selected actions for aggressive, criminal and normative actions.

[78]The possibility to have a general reusable normative reasoning (software) component was discussed in GLODERS but in the end not realised. However, this still is an imaginable option for a model extension.

Aggressive Actions Aggressive actions are the concrete means by which aggressions are expressed. Aggressive actions can be distinguished in causing harm either in a physical or in a mental manner. The categories of aggressive actions defined as (sub-) objects in the conceptual model are extracted from codes identified in the qualitative data analysis, but can be considered a valid abstraction on a more general level. The instances selected for the different categories are introduced in the following, with their anticipated strength (in the sense of severity) and frequency as reported in the evidence documents.

Violent aggressive actions always cause physical, sometimes also mental harm. In general it can be assumed that violence causes more severe harm than other kinds of aggressive actions. Two cases are regarded in the model:

- 'Beating up', modest strength, reported in three cases, e.g.:

  > "Based upon witness reports, in the night from 31. December [...] to 1. January [...] V2 has been thrashed soundly by O1 and others."[79]

- 'Attempting to kill', high strength, reported in 18 cases, e.g.:

  > "Attempt to liquidate B24."

  > "On Monday, 17th May [...] V01 had been shot to death at YYY. "

Mostly mental harm is caused by a category of aggressive actions can be subsumed under the term threat. The 'most popular' ways to apply threat to other criminals are:

- 'outburst of rage', low strength, reported in six cases, e.g.:

  > "For the first time I saw that he had lather in his mouth and kicked a bicycle against a tree."

- 'death threat', high strength, reported in eleven cases, e.g.:

  > "It was very tough at that times because he was threatened. He said that 'they' wanted to kill him. K. then he said that 'they' would not kill the hen with golden eggs."

  > "He said that his children had been in great danger."

- 'kidnapping', low strength, reported in three cases, e.g.:

  > "O1 tried to kidnap me."

[79] This and the following citations are (translated and anonymised) phrases from the evidence documents and can be consulted in annotations of the respective CCD elements.

- 'having an eye on you', low strength, reported in nine cases, e.g.:

> "O1 had V01 in his grip. He shall do as told, otherwise his family would have a problem."

- 'putting a gun into the stomach', high strength, reported in one case (but with extraordinarily high impact), e.g.:

> "At the point at which you get pressed a machine gun in your stomach."

- 'demanding money in dark forest at midnight', modest strength, reported in four cases where the victim of the threat was the White Collar criminal, e.g.:

> "At May 15, [...], O5 came to my house in order to say that [...] in the evening I should come to the forest. This is standard: intimidate and request for money."

The number and kind of instances selected here are much more illustrative than e.g. for violence as they express very well the character of the particular criminal network under investigation, making the model and simulation results comprehensible to the stakeholders.

The rationale is similar for the selection of instances for the two ways of betrayal, the third kind of aggressive action. In the case of internal betrayal three cases are included:

- 'stealing capital (drugs or money)', high strength, reported in one case (but with an anticipated higher dark figure), e.g.:

> "Following O1 C. betrayed him [...]."

- 'starting affair with girlfriend', low strength, reported in two cases (also with an anticipated higher dark figure), e.g.:

> "I believe [...] V01 started an affair with N. (B19)."

- the special action of 'refuse to return entrusted money', is of high strength, but only applies to the case where White Collar criminal does not fulfil his commitment to ensure smooth money laundering operation (for whatever reason):

> "Earlier this month I paid [...] to you. This was an instalment of the back-payment to Mrs. G and B29 which have been awarded to your client. There is a considerable backlog demand in the back-payment. The reason is twofold: first, it's becoming difficult to gain new funding because of the negative reports in the media and second much of our liquidity has been lost in payments to O1."

External betrayal is represented by two cases:

- 'Going to police', only modest strength since it is not visible to the other members of the criminal network in the first place, but might have much bigger impact via follow-up events triggered by police intervention. Reported in two cases, e.g.:

  > "At Aug. [...] the criminal intelligence of the regional police received an anonymous letter."

- 'Going to public', high strength (in fact the worst violation on the norm of trust), reported in four cases, e.g.:

  > "[...] I think it was in the times when the pictures of him and O1 had been published."

As indicated before, the main guiding strategy for selecting the instances of aggressive actions is to cover the most characteristic examples for the criminal network. Referencing these examples in the outputs generated by the simulation model later on improves the readability of these automatically generated logs. The anticipation is that the simulation log tells a story that can be read and further analysed and condensed by human readers with less 'pain' than an abstract notion of aggressions might cause. This has, however, implications on the style the rules have to be implemented: The decision processes regarding aggressive actions need to be two-staged. In the first stage the category and strength of the aggression is decided, if possible in a deterministic manner. This means, whenever the evidence gives enough clue on which conditions which kind of aggression is most likely to be pursued, this branch is selected. On ambiguous cases either cognitive heuristics come into play, and/or a recourse to stochastic processes. Hence, this stage shapes the actual logic of the model. The second stage is then the selection of the concrete example, predominantly by randomly selecting the instance, taking the frequency (or rather the derived probability) of the options into account. This approach for implementing decision processes will be illustrated in the following Chapter 6, e.g. in Figure 6.6.

Criminal Actions The criminal actions encompass situations in which criminals engage in course of their ordinary business. These are typically not relevant in situations of conflict among criminals, but irregularities as they might occur in any kind of business, and which can have quite dramatic consequences on the stability of a criminal network. Since details of the everyday business are not relevant for the model subject, just one object drugTrafficking with a related instance 'generating income from drug trafficking' are incorporated in the conceptual model as a kind of placeholder.

The other criminal actions regard the — more relevant — money laundering for which two objects are present. The first object is requestForLaunderedMoney, an action performed by a Black Collar criminal targeting the White

Collar where the laundered and now legal money is requested. Here two instances are defined:

- 'request for invested money — low': The Black Collar Criminal asks for a small amount of money that should be easily payable by the White Collar. 'Small amount' is arbitrarily set to 6,000,000 units of a fictive currency. This is reported e.g. in the following annotation:

  > "At the request of Mr B7 B9 appeared at his office in [...]. There he was forced to sign an irrevocable certificate of authority."

- 'request for invested money — high': The Black Collar Criminal asks for very large amount of money that can bring the White Collar in trouble (at least in case of multiple requests) since e.g. capital might be bound in long-term investments . 'Large amount' is set to 20,000,000 units of a fictive currency. Examples for annotations are:

  > "At the beginning of Oct. [...] S.K. came in the office. She told the employees that she needed to talk to me because her former man (who died) had 7 millions active debts. If I don't pay her [...] friend O6 would kill me."

The second object is acceptToReturnLaunderedMoney, the answer of the White Collar criminal to a request by the Black Collar. This answer can be positive or negative, hence two instances are defined:

- 'accept to return laundered money', rendering a successful transaction, for example:

  > "V01 has finally paid 59 millions. [...]"

- 'refuse to return laundered money'. the unsuccessful case, at the same time constituting a norm violation. An example in the evidence is expressed in the following annotation:

  > "At a certain point he had problems with his liquidity."

The 'run on the bank' modelled in 5.4.5 makes use of these two objects; in fact all actions involved in this process are covered by the four related instances. As shown in the action diagram, in case of a refusal to return laundered money, also aggressive actions become relevant.

Normative Actions The normative actions are an abstraction meant to cover the manifestation of norm in the behaviour of actors. Within the modelled normative system of criminal agents the moral norm 'Commitment to norm of trust in organisation' (NORM(1) in equation 5.1) is the 'master' norm, and hence used as reference point for the normative actions. The other

norms are not accompanied with normative actions (for the model version at hand), but will influence the simulation model in an informative way.

For the three sub-objects of normative action, exactly one instance is defined:

- 'enforce norm of trust (general)' for the object normEnforcement. Norm enforcement is expressed by aggressive actions, so that the annotations mentioned there apply also here as well as for:
- 'violate norm of trust (general)' for the object normViolation. The interpretation whether an aggression is a norm enforcement or violation is one of the major research questions for the model.
- 'obey to norm of trust (general)' for the object normObedience. The following annotation provides evidence for this instance:

> "I guess that B also fears O1. Why do you think so? Because he is always willing for O1. [...]"

5.5.3 Agent architecture and decision processes

As pointed out previously, the conceptual model described in the previous sections was developed with the initial aim to find a common language and understanding between the persons involved in modelling with background in social sciences and computational science, and to have a medium to communicate and discuss the modelling artefacts with police stakeholders. Later in the conceptual modelling phase (i.e. starting with the second iteration, still not having an 'inter-coder reliability' fully reached; see qualitative analysis process in section 4.4.1, Figure 4.3), also more 'technical' entities were included in the CCD to clarify vague concepts and to show possible ways of implementation. The CCD approach turned out to be specially effective for these purposes, and it also promised to be capable to level up the model specification towards a more implementation-directed shape, by adding purely technically motivated concepts. However, the intention remained to — as far as possible — keep the CCD free from those concepts without any relation to the evidence base or topics pointed out by stakeholders, or without a theoretical backing in social science or criminology.

This section starts at the point where the state of 'inter-coder reliability' was finally agreed. This agreement triggered a design process focussing on technical details of the simulation model, in particular the agent architecture and decision processes. The resulting artefacts are flow charts[80] for specific parts of the action diagram, where due to missing hints in the evidence the conceptual description is too abstract to be reasonably formalised in an implementation.

[80]Flow charts were used to be aligned with the other use case model developed in the GLODERS project; see (Nardin et al., 2016a). For example, UML activity diagrams could also have been used.

While in context of the conceptual model in CCD terminology the *actor* refers to humans with cognitive and behavioural abilities, in the technical specification presented in the following subsections the term *agent* is used. The agent is therewith an entity projecting these human properties into a technical representation, applying reasonable abstractions (see also section 2.2.3). Earlier versions of these intra-agent decision processes have been presented in a deliverable of the GLODERS project (Andrighetto et al., 2014).

Action '`perform aggressive actions against member X`'

The normative reasoning sketched in the flow chart in Figure 5.19 refines the action `perform aggressive actions against member X` in the action diagram part shown in Figure 5.3. It covers the norm enforcement sub-process, one of two cases where normative reasoning comes into play.

The normative reasoning process sets the idea of a generic normative process discussed and conceptualised in GLODERS (which is further elaborated in section 5.5.4) in connection with the agent architecture of the criminal actor. The normative process can be triggered by two different events:

- In the case relevant here by the external event `Member becomes disreputable`. This is an event concerning another agent, e.g. the observation that a member of the criminal network is becoming 'too greedy' — in accordance with the condition in the related action diagram.

- The other case (here greyed out) is the internal event `Possible normative motivated aggression recognised`, further detailed below.

After a pre-processing and *classification* of the event, the information is passed to the *normative process*, which can be regarded as a black box at this point. The expected outcome is a decision whether or not a sanction should be issued towards the disreputable member, in form of some kind of *aggression* to be selected in subsequent decision.

Action '`member X interprets aggressive action`'

The action `member X interprets aggressive action` as tie point of the two action digram parts in Figure 5.3 and Figure 5.4 is mapped out in several flow charts shown in Figure 5.20 to Figure 5.23.

The intra-agent process of interpreting an attack starts with a reasoning about the experienced aggression. This process is outlined in Figure 5.20. It is triggered when the agent recognises an *aggression against itself*, and comprises the first three stages of the decision process leading to possible reactions on the aggression.

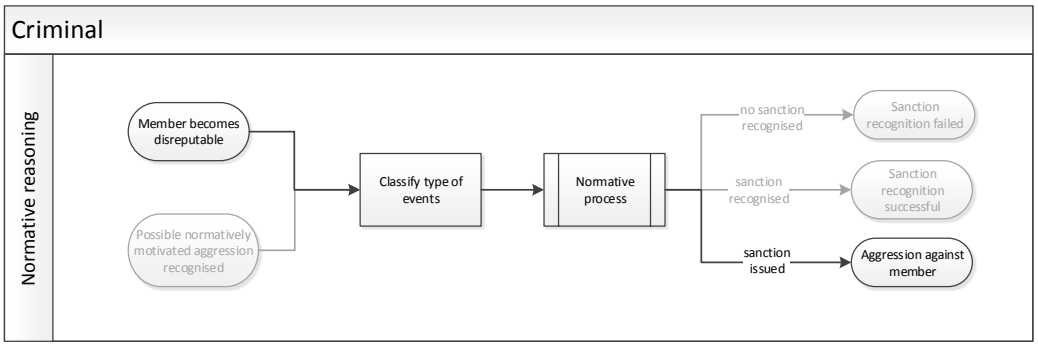

Figure 5.19: Flow chart of the normative reasoning following the event that a member X becomes disreputable

First stage In the first stage (exhibited in Figure 5.20) it is assessed whether the *aggressor is reputable* and whether there is a motivation for the aggression. Information on trustworthiness of the aggressor is taken from an *image and reputation repository*. This reputation sub-system is discussed together with the normative process in section 5.5.4. If the aggressor is reputable, a *possibly normatively motivated aggression* is anticipated and the normative reasoning process is triggered at the second stage (see Figure 5.21 and next paragraph). If no sanction is recognised by the normative process (i.e. the sanction recognition failed), the agent has to decide whether the aggression is serious enough to pose a potential threat to itself. In this case, and in the case that the aggressor is recognised as not reputable, reactions will be triggered by entering the third stage of the process in which the 'operation mode' of the agent is either set to *a rational or an emotional frame*. The actual switching into one of the two frames is done in separate processes, sketched in Figure 5.22 (and detailed in the next but one paragraph).

Second stage The second stage of the reasoning about aggression uses the same *normative reasoning process* as introduced for the norm enforcement sub-process in the first paragraph of this subsection. This time the sanction recognition sub-process of the normative process is regarded and shown in Figure 5.21; the triggering event is that a *possibly normatively motivated aggression is recognised*. The normative process can either reveal that the aggression is most likely *no sanction*, i.e. no norm could be identified which links to the triggering event. This might lead to defensive aggressive reactions as stated above. On the other hand, the result can be a *recognised sanction*. In this case the agent decides whether to obey the sanction and to strengthen the commitment in the norm of trust, or it decides to cheat in order to e.g. maximise individual profits at the expense of fellow criminals (and subsequently perform

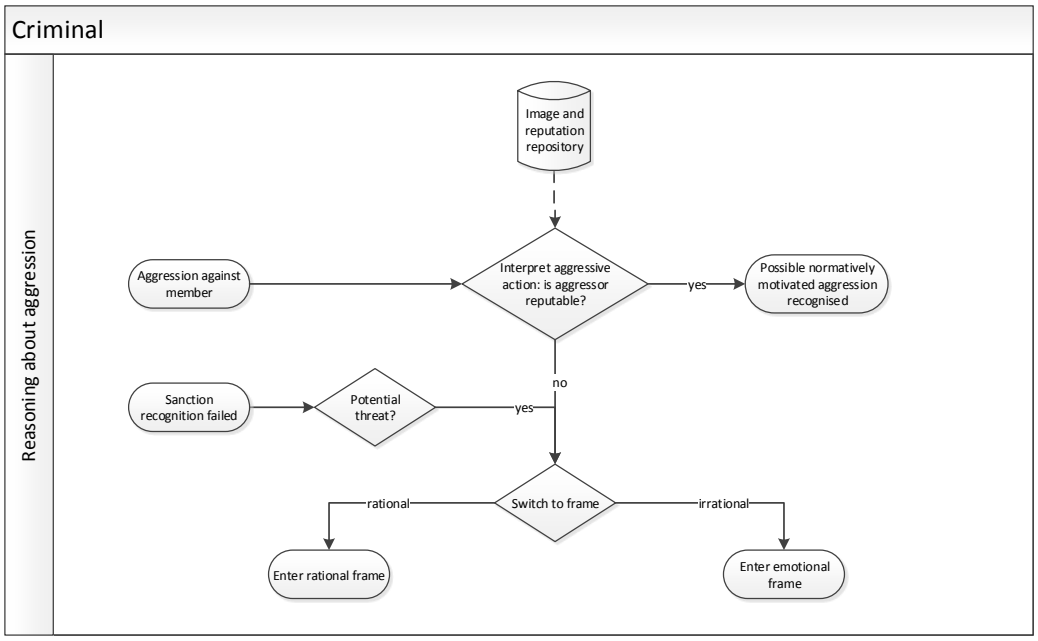

Figure 5.20: Flow chart of a criminal's reasoning about an experienced aggression

actions like betrayal of the organisation). Hence, a *balance of a normative drive and an individual drive* of the agent has to be established (basically a decision between social responsibility and selfishness). These different drives are related to the concept defined and used in (Andrighetto et al., 2013).

Third stage As indicated above, in the third stage of the reasoning about aggression a decisions is made about the mental frame of the agent. This synthetic decision resembles a situation in which a human is confronted with a threatening or otherwise dangerous event that might exceed the cognitive capabilities of the individual to be handled in a rational way, e.g. by reasoning about options for solutions to minimise the expected harm. In this case, emotions become the driving force for actions, for which possible negative consequences in the long-term perspective are no longer considered. The abstraction and technical realisation is covered in the flow charts in Figure 5.22. Technically, attributes of the agent are configured that inform subsequent decision processes and, thus, biasing the behaviour of the agent. The sub-processes described subsequently rely on these attributes.

Reaction on aggression The final stage of the interpretation of the aggressive action is the first decision stage on an adequate reaction. The flow chart

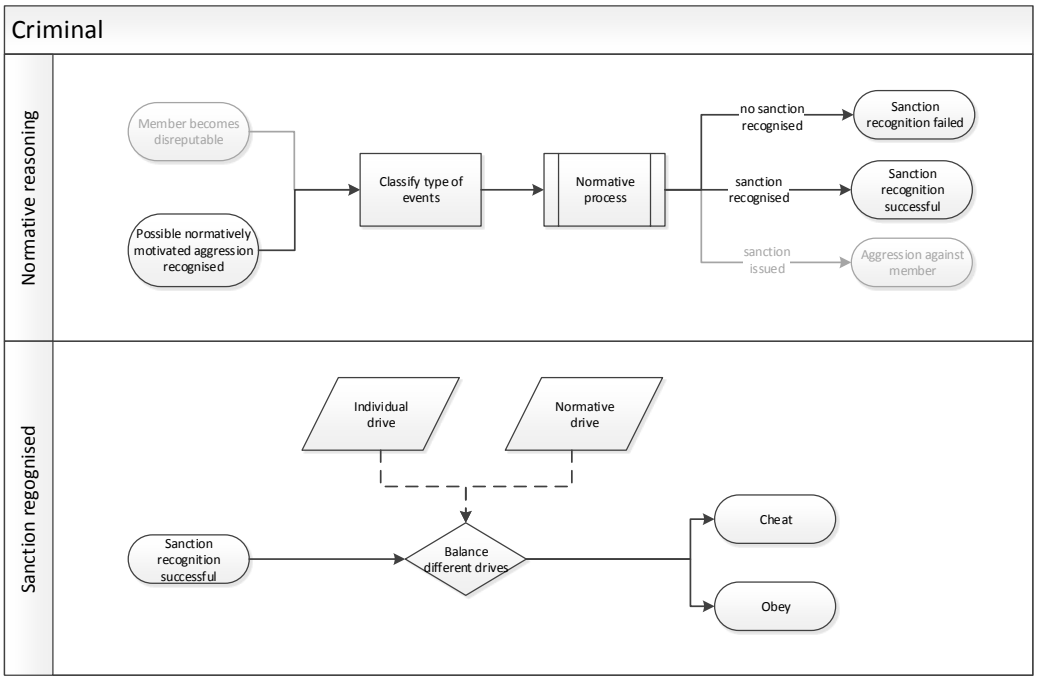

Figure 5.21: Flow chart of the normative reasoning as part of reasoning about an experienced aggression

in Figure 5.23 shows the two possible options for a reaction: either to *perform a counteraggression*[81] or to *betray the criminal organisation* (these two options are detailed in the following subsections). A decision for a reaction is considered following the entering of either the *rational* or the *emotional* mental frame, or if the agent decides to *cheat*. The distinction of these three conditions is explicitly modelled here since they constitute one of the parameters for the decision, together with *current mental frame of the agent*.

Action '`member X performs counteraggression`'

If the interpretation of an aggression concludes with the decision to react with a counteraggression (i.e. a violent or threatening action), the concrete aggressive action has to be decided in another decision process and finally executed. This is hidden in the action `member X performs counteraggression` in Figure 5.4. Figure 5.24 unravels this action a bit further.

[81] A note about the terminology used: Counteraggression in this context is restricted to violent and threatening aggressions and separated from betrayal. For the initial aggression this distinction is not made: violence, threat and betrayal are all embraced in the term aggression.

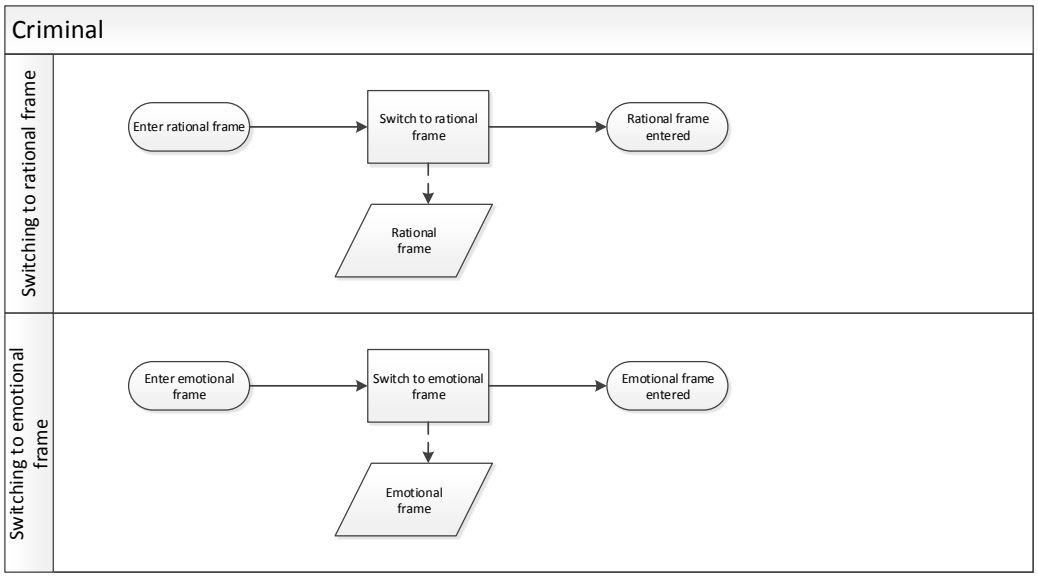

Figure 5.22: Flow charts for switching into the two mental frames — rational or emotional

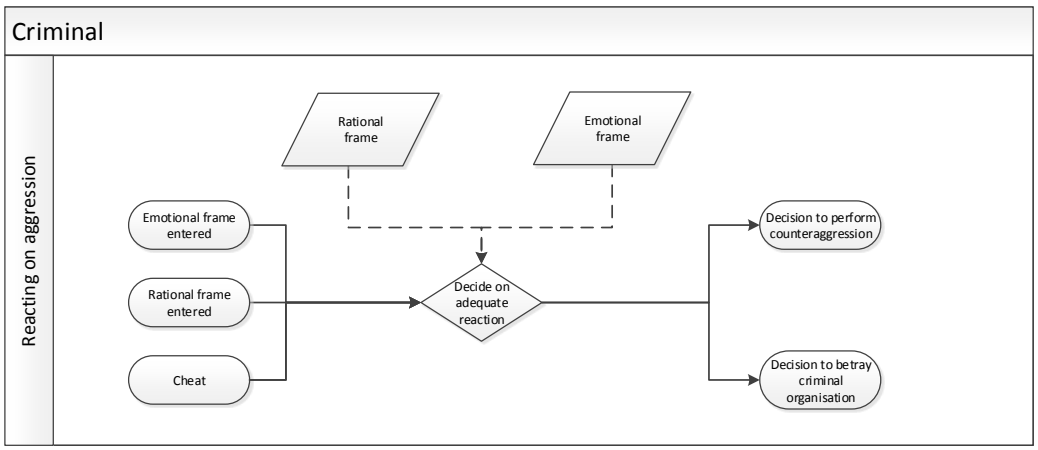

Figure 5.23: Flow chart of a criminal's decision on the kind of reaction on an experienced aggression

Firstly, the *members responsible for the attack have to be identified.* In the trivial case the responsible member is identical with the aggressor and can be identified in the act of aggression. However, rationale for having this

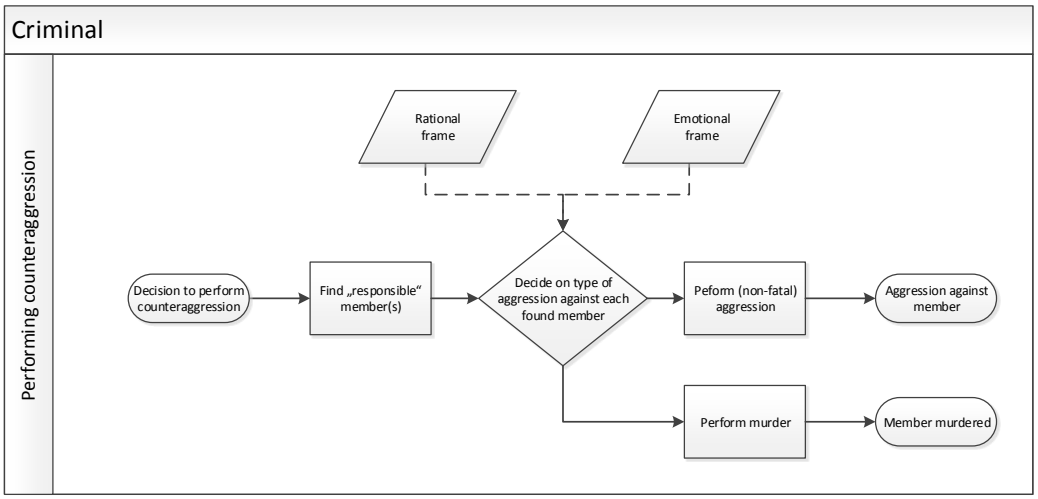

Figure 5.24: Flow chart of a criminal's decision on the kind of counteraggression

search process included is to cater for special cases where the aggressor remains unknown (e.g. in case of an anonymous criminal complaint), or where more than one aggressors are involved (e.g. the instigators of the aggression).

Subsequently the *aggressive actions against the aggressor(s) can be decided*. This decision is made on base of the severity of the initial aggression and strongly influenced by the *current mental frame* of the deciding agent. Hence the decision is made either in a rational decision process or by an emotionally driven panic reaction. As result of this decision just two options are distinguished here: an *aggression with non-fatal intention*, or a *murder attempt*.

Action '`member X decides to betray criminal organisation`'

The decision process for betrayal (as the other way to respond to an aggression) is already quite elaborated in the action diagram part in Figure 5.5 as an extension of the interpretation process in Figure 5.4, with the action `member X decides to betray criminal organisation` as connecting point. The flow chart in Figure 5.25 complements the action diagram by defining a sequence of decisions: First a *decision is made whether the betrayal should be internal*, i.e. only involving the original aggressor and the victim (now acting as the aggressor). Such a typically *'nasty' action* causes the latter to *become disreputable*. If the agent decides against an internal betrayal, a second decision between two more options for *external betrayal* has to be made (in alignment with the action diagram): either to *go to the police* or to *inform the public*. Both decisions are biased by the *mental frame* of the deciding agent.

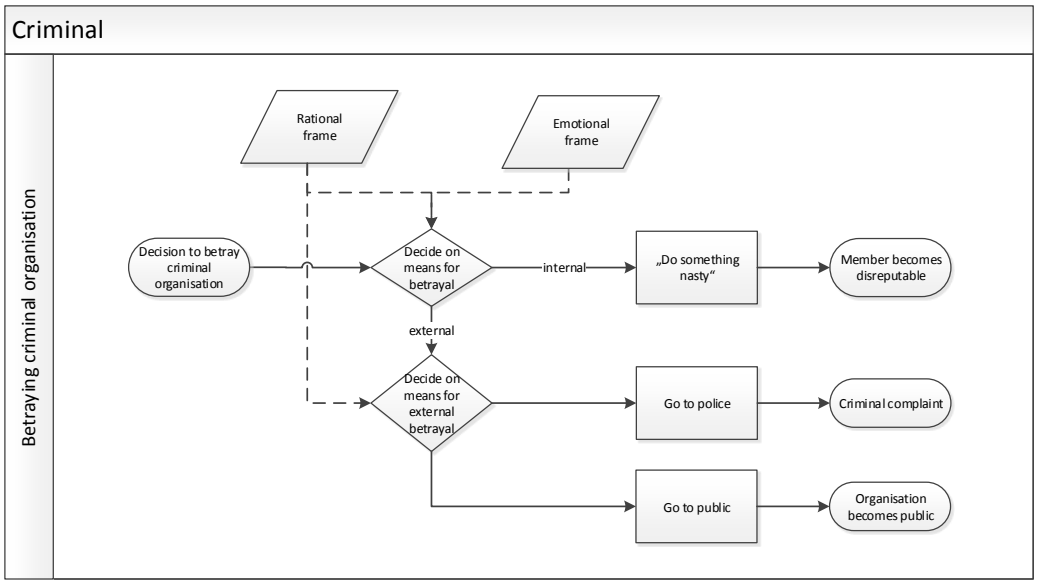

Figure 5.25: Flow chart of a criminal's decision on the kind of betrayal

Concept for evaluation of trust

An inherent cognitive process not underpinned by the evidence from police interrogations is the evaluation of the state of trust within the criminal network, as shown in the action diagram part in Figure 5.9. This is a very rough abstraction of psychological mechanisms to represent and combine rational aspects with emotions[82] for an estimation of (or a 'feeling' about) the cohesion of the network members in supporting the collective goal of the group. This abstraction is based on the assumption that a certain level of trust between the network members is needed to assure stability. The level of trust is a subjective interpretation of the situation.

According to the action diagram in Figure 5.9, the internal state within the criminal group is evaluated by each member, if violent aggressions against other members (e.g. murder) or possibly other actions that violate the norm of trust (e.g. revealing the network to the public) are observed, but also if a criminal obeys to a sanction, or observes a fellow criminal obeying. This is similarly represented by the four trigger events in Figure 5.26. This flow chart captures details beyond the action diagram together with first thoughts for possible implementations.

First of all, this process has to be considered in the context of the reputation process (see section 5.5.4), as initially the *image and reputation repository* is

[82] A recent survey of work related to modelling emotions in social simulation is provided by Bourgais et al. (2018).

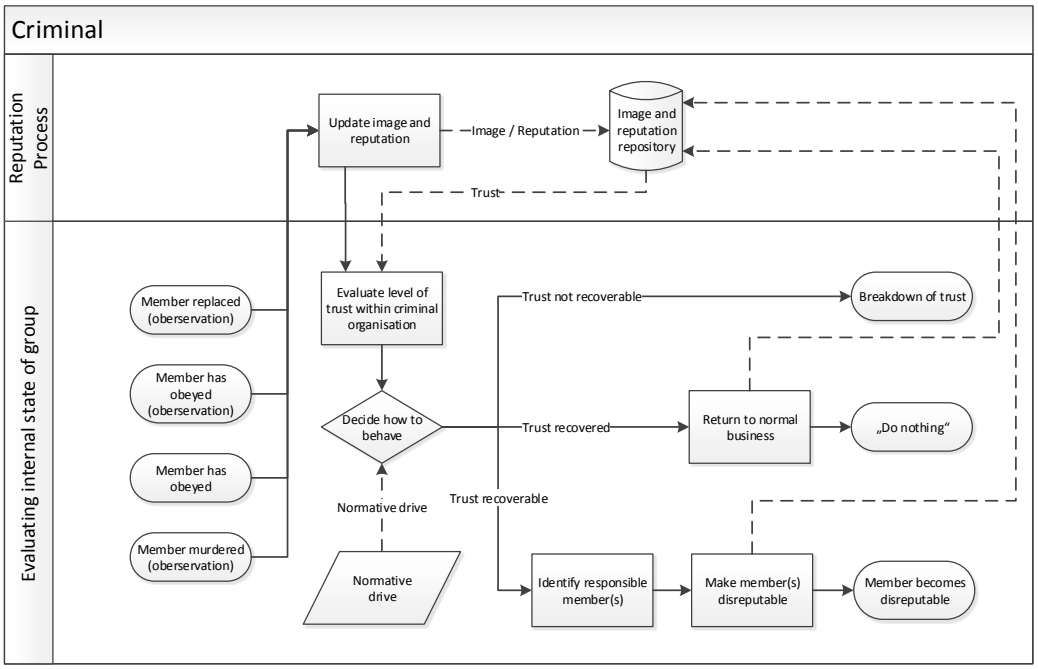

Figure 5.26: Flow chart of a criminal's evaluation of the state of trust within the network

updated, a database reflecting the individual image values for all other criminal agents and the global reputation values of all criminal agents. The updated values are used to 'calculate' and *evaluate the trust level*. According to the trust level, the further *behaviour of the agent is then decided* with consideration of a *normative drive* as a parameter, i.e. the incentive of the agent to follow the norms of the network. Three options for directing the behaviour are differentiated, as already indicated in subsection Concept of Trust (and initially elaborated in (Andrighetto et al., 2014)):

- The result of the evaluation is that the trust has been *recovered*; the agent returns to 'ordinary business' and breaks the cycle of aggression.

- The trust seems *recoverable* if certain measures (like sanctioning) are taken. In order to recover the trust, the *disreputable members have to be identified* in order to enable the normative process also at other agents, i.e. to trigger sanctioning actions by other criminals.

- The trust among the group members is *corrupted* in a way that it cannot be recovered, leading to total *breakdown of trust*, which possibly triggers panic in several or all the agents.

5.5.4 Normative process

The normative process used for this use case model is based on the EMIL-A normative architecture. A first version of this architecture was developed in the EMIL project[83], on the basis of a normative theory described in (Conte et al., 2013). This architecture was advanced and adapted for the purposes of the use case models within the GLODERS project. Nardin et al. (2016a) show the usage of this architecture for the GLODERS ERS use case model (see 4.2); in addition, an older but more detailed description can be found in (Andrighetto et al., 2014).

A flow chart of the normative process derived from the EMIL-A normative architecture, amended for the use case introduced in this chapter is presented in Figure 5.27. In alignment with Figure 5.19 and Figure 5.21, the starting point for this normative process is a *classified event*, an event with attached normative (meta) data, i.e. about type, severity and originator of the event in relation with possibly regarded norms. In this model, two events play a role:

- The event that a *member becomes disreputable* which transports information about a norm violation. The attached meta-information regards reason and strength of the misdemeanour.
- The event of a *recognition of a possibly normative motivated aggression*. The meta-information specifies the aggressive action and aggressor in more detail.

The expected outputs of the normative process are on the one hand the information whether or not an event is recognised as a sanction (or more general as a normatively motivated action), on the other hand a sanction issued as a reaction on another agent's norm violation. The normative process basically works in a way described in the following subsection.

Operating principle of the normative process

According to Figure 5.27, the classified event is processed by `Update normative information`, from where the `Normative information repository` is updated. There all types of events (such as obey to a norm or violation of a norm) are stored. With this updated normative information the norm salience can be updated (`Update norm salience`) and stored in the `Normative Board`, which contains all known norms and sanctions. By the norm salience "the importance of the norm, which plays a major role in that norm's acceptance or rejection"(Nardin et al., 2016a, p. 1129) is expressed. Updating the norm salience might also trigger `Norm adoption`, which takes norm beliefs from the `Normative Board` and writes back norm goals. "The norm adoption module allows agents to decide whether to adopt or not the identified social norms as normative goals"(Nardin et al., 2016a, p. 1128).

[83] `http://emil.istc.cnr.it`

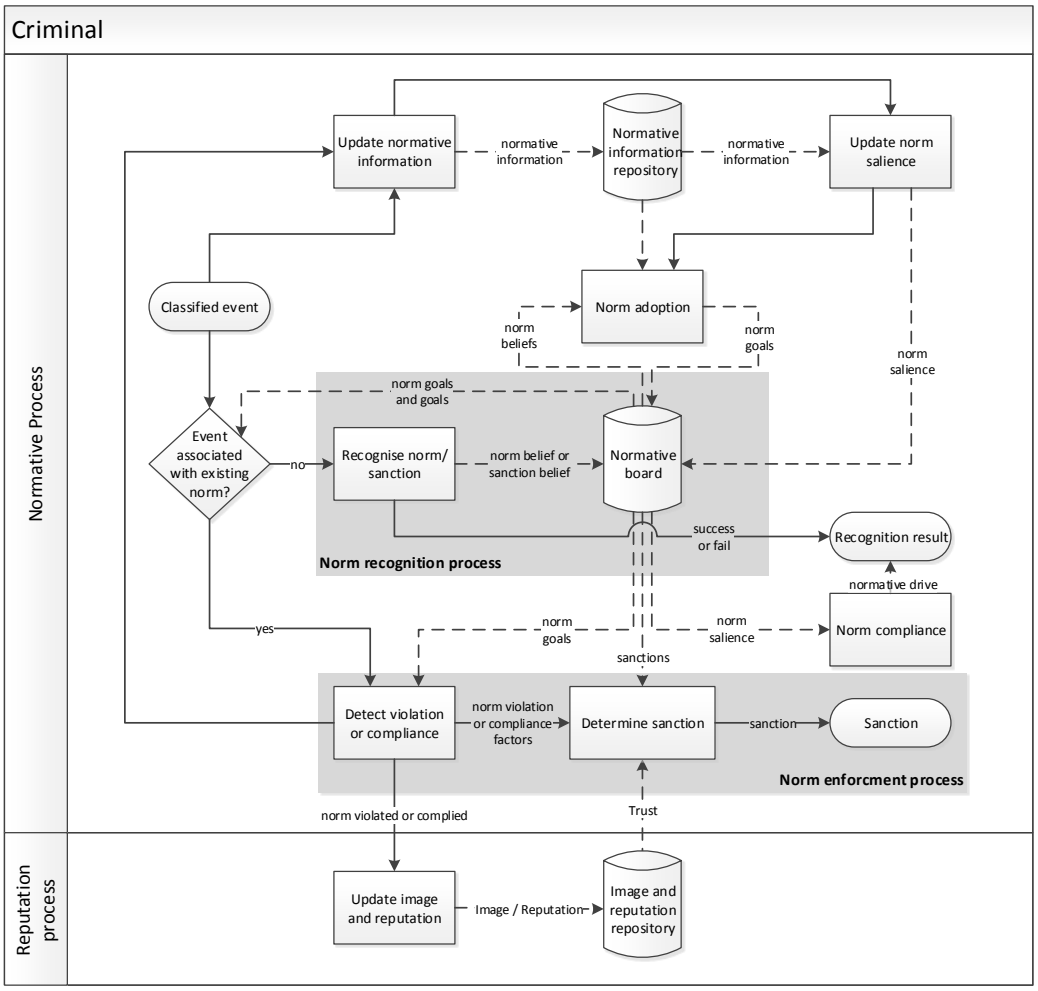

Figure 5.27: Flow chart of the adapted EMIL-A normative process (slightly modified from Andrighetto et al. (2014))

The classified event is also confronted with an evaluation whether the *event is associated with an existing norm*, i.e. a decision if a known norm unambiguously relates to the event. For this decision the norm beliefs and goals from the `Normative board` are consulted. If no norm is immediately associated with the event, a `Recognise norm/sanction` process is triggered resulting either in success or failure as `Recognition result`. This information is written as norm belief or sanction belief in the `Normative board`. With this recognition process "agents infer new social norms, and generate the corresponding normative beliefs, through observations and interactions with other agents"(Nardin et al., 2016a, p. 1128). The result can also be complemented with a normative drive information, determined by `Norm compliance`, taking the norm salience

stored in the `Normative board` into consideration. "The `Norm Compliance` module determines whether or not to have the intent to comply with the adopted normative goals and convert them into normative intentions" (Nardin et al., 2016a, p. 1128).

If the event is associated with an existing norm, the `Detect violation or compliance` process can be started. Using information from the `Normative board`, an `Update normative information` (with the resulting sub-process as sketched above) might be activated with new information on the norm violation or compliance, as well as `Update image and reputation`, bringing the information in `Image and reputation repository` up to date. With the updated `Normative board` and `Image and reputation repository` the `Determine sanction` process is triggered with norm violation or compliance factors in order to determine an appropriate sanction as result of the norm enforcement.

Realisation of the normative process

To take a look ahead, the model implementation does not rely on a centralised normative component. While the original plan was to create a common software building block implementing the normative process, it turned out that the technical requirements (with regard to model architecture and formalisation of the norms) were so different that the complexity of realising such a component would have exceeded the available resources in the GLODERS project (and possibly jeopardised the commitment to deliver productive simulation models). Hence, the concept of the normative process can be considered an architectural building block. The single sub-processes of the normative architecture are 'spread' over various parts of the model with dedicated declarative rules. In the following, a couple of examples of the normative reasoning (taken from (Andrighetto et al., 2014, p. 48–49)) are drawn, highlighting the correlation of model-specific details to the architecture components introduced before:

a) "The salience of the norm 'distribute return of investment' is enhanced by intimidation of the white collar criminal: 'I paid but I'm alive', means that he obeyed to the norm because of (severe) norm enforcement; i.e., intimidation leads to an update to the normative information in the intimidated person. However, note that an enforcement of the behavioural rule (norm) might at the same time decrease the cognitive commitment to the organisation (value)."

b) "Individuals might be tempted to actively enforce a certain norm once they detect norm violation. There exist several instances when criminal network members went to police, but the criminal network got notice of that. These criminal network members have been killed to enforce

the norm 'commitment to the norms of trust in the organisation.' Passing information to the police is the most severe violation of this norm, resulting in death penalty."

"Additionally the normative process triggers two processes out of the abovementioned normative sub-processes a) and b)."

1. "Deliberately disobey a norm. Criminals might be tempted to cheat. This is part of the norms salience process by reducing the salience. There was a struggle between one member of the criminal network and a foreign one. It is unclear whether the criminal network member wanted to cheat the foreigners or whether this was by accident. However, the criminal network member was killed for that reason (death penalty)."

2. "Defence: The guy who had to endure a machine gun in his stomach tried to murder (without success) his former friend in return. The situation had been part of the attempt to extort the white collar criminal in order to get the invested money back (run on the bank). This is an enforcement of the norm 'Distribute return of investment'. However, he failed to recognise the attack on his life as a sanction. Anybody who is in fear of life (which might be interpreted as death penalty) might try to save his life. Counter-aggression might be a suitable means."

The actual implementation furthermore focusses mainly on sanction recognition. The variant described in this PhD thesis is restricted to the top-level moral norm 'Commitment to norm of trust in organisation', reflected in image and reputation values. Obligations are coded in rules, e.g. to generate normative events if obligations are not followed, like money is not paid back by the White Collar criminal, or arbitrary excessive violence is used. Further details on the implementation are given in the following Chapter 6.

5.6 Wrapping up the conceptual modelling: Requirements for the simulation model

The explanations presented in this chapter contain — partly implicitly — a comprehensive set of requirements for the envisaged simulation model. Even though the purpose of the conceptual model — the CCD — is mainly to provide a means for discussing the model subject with police stakeholders and domain experts, with less priority to 'easy' transformation, substantial parts of the model program code can be generated automatically from the CCD. With the CCD2DRAMS tool as part of the OCOPOMO toolbox,

- the actors are automatically transformed into agent classes,
- the objects into data structures for DRAMS (i.e. fact templates),

- the instances into facts, and
- the actions are the basis for rules, with templates indicating which facts constitute the conditions.

However, the main effort for the SIMULATION MODELLER remains to 'breathe life' into the rules. This includes making the model run in an consistent and correct (in terms of sequence of events) way and producing outputs that can be read by humans and also further analysed with appropriate tools. These activities introduce additional aspects, which have to be formulated as requirements:

- The simulation model uses stochastic processes to some extent (hence, it is not a purely 'deterministic' model), in particular when it comes to decisions for which sufficient information does not exist in the evidence. Here, random number generators play a central role, at least as an interim solution. As soon as further data would become available, these decisions could be made deterministic, or in any other way better informed.
- A decision about the temporal structure of simulation runs has to be made: The regular ticks supplied by the timer of the simulation framework are superimposed with a kind of event-driven approach, where the time period between two ticks is not specified. Hence, the sequence of consecutive ticks represents progressing time, and each tick marks a point of time where scheduled events are executed. However, introducing a 'processing time' is considered by deliberately postponing events and re-scheduling them to some subsequent tick, although the factual conditions are fulfilled previously. This is relevant e.g. for police operations, which usually are executed a considerable time after the start of investigations, while the criminal activities of the suspects continue (and might deliver further evidence to the police).
- It is important to carefully and extensively add annotations to the elements of the conceptual model, so that the outputs generated by the rules contain the respective traceability information necessary to apply methods for simulation result analysis and presentation as projected for GLODERS (Chapter 4) and initially proposed by the OCOPOMO process (see section 3.4).

Before going over to the implementation in the next chapter a few words to conclude the conceptual modelling:

The plethora of information presented in this chapter is contained in a single conceptual model (with exception of the flow charts of the implementation directives for the decision processes). The structure and parts of the conceptual model allow to keep track of the content and to get different views on the model — from overview graphics to expert annotations for single elements. The extensive documentation of the model given here (although it rather has

the purpose of a demonstration) could also be added to the model in form of annotation, i.e. to make the conceptual model 'self-documenting'.

The elements included in the conceptual model should not constrain but rather guide the implementation, so that the implementer has a reasonable degree of freedom. This will be evident in the simulation model documentation presented in the next chapter, where the concepts influence the formal model in different ways. So are, for example, some of the objects (together with object instances) realised by data elements (or facts in a rule engine environment), while other objects just shape the construction of rules, e.g. the formalisation of conditions or the text output of a rule.

Chapter 6

Simulating internal dynamics of a criminal network

6.1 Introduction

This chapter carries further the results from Chapter 5, where the qualitative analysis of texts from police investigations about a criminal network is transformed into a conceptual model of the dynamics that led to the violent breakdown of the criminal network. This conceptual model is transformed (as described in section 3.3.3) and formalised into a simulation model, which is subject of this chapter. An compressed version of the content presented here has been published before in (Lotzmann and Neumann, 2016) and carried further in other publications, e.g. by Van Putten and Neumann (2018).

The chapter is structured as follows: After a short excursion to the applied simulation approach and environment, the subsequent section gives an overview on the simulation model both in terms of static and dynamic aspects. The former includes the agents and related attributes of which the model is comprised, the latter includes the control flow in the different parts of the model. In the remaining sections, this control flow is further detailed in order to give profound insights on concrete design decisions to show how the evidence is reflected in the implementation and to facilitate replicability.

6.2 Simulation approach and environment

The target platform of the applied model-to-code transformation is the simulation framework Repast in conjunction with the declarative rule engine DRAMS. The application of DRAMS as simulation tool shapes the implementation style in a particular direction: The entire agent behaviour is specified by rules in a declarative programming language, processing the knowledge stored as facts in so-called fact bases. The concept of rule processing in DRAMS is described in depth in Chapter 8. The following paragraph provides a brief overview to make the content of this chapter comprehensible.

As DRAMS is designed as a distributed rule engine, each agent is equipped with its own fact base and own rules, while for 'world knowledge' and also communication purposes a global fact base is provided. Each rule consists of a condition part, the Left Hand Side (LHS) and an action part, the Right Hand Side (RHS). The conditions in the LHS are specified using a set of clauses e.g. for performing factbase queries, binding variables, comparing variables and constants, doing mathematical calculations and so on. The RHS consists of clauses that allow for modifications of fact bases (asserting new facts, retracting existing facts) as well as clauses for writing simulation outcomes in different ways. The basic mechanism of the rule engine is then to evaluate (in a pre-calculated schedule based on data-rule-dependency graphs) the LHS of all rules for which all conditions are fulfilled, and afterwards fire these rule by executing the RHS. The new facts generated by the RHS are setting the condition for new rules to fire.

The actual implementation of the simulation model is shaped by the conceptual model, not at least due to the applied code generation. All the actions modelled in the CCD action diagram are also present as DRAMS rules in the simulation model. In order to achieve a consistent implementation, a number of aspects are added to the model which are not described in the evidence base, instead relying on cognitive heuristics (see section 3.3.4). On the other hand, some details included in the conceptual model are left out to keep the complexity of the simulation model manageable, but also due to decisions to concentrate the focus on some crucial aspects of interest for deeper analysis of the case.

Main reason to ground the simulation model on DRAMS is the opportunity to benefit from the traceability functionality built in the OCOPOMO toolbox. Herewith it becomes possible to trace simulation results back to the phrases from the evidence base annotated to elements of the conceptual model. I.e. this functionality opens a way to efficiently perform qualitative analysis of simulation results by means of unveiling the relations between dynamics in simulations runs and events in the real criminal network described in the evidence base.

6.3 Simulation model overview

In the simulation model, agents are included for the CCD actor types Black Collar Criminal, White Collar Criminal and Police. While for the two types of criminals arbitrary numbers of instances can be created for simulation runs, the police is represented as an institutional agent, i.e. a single agent instance covers the activities of this actor. In typical simulation runs there exists a single White Collar Criminal, which is responsible for money laundering and is typically also part of the legal world, but might become involved in aggressive practices of the Black Collar Criminals. These are the actual representatives of the illegal world of the criminal network. There are two types of Black Collars

distinguished, one called the Reputable Criminal which is initially in the so-called rational mental frame, while the other 'normal' Criminal only acts in the emotional mental frame. In the course of the simulation the Reputable Criminal, too, might switch to the emotional frame e.g. due to violent events. This distinction between the two types of mental frames — ad discussed in the previous chapter — is illustrated below.

As mentioned before, the model is implemented in a discrete event manner where the course of time is represented by ticks, but where no concrete time period between ticks is specified. Actions or reactions involving multi-staged decision processes are typically spread across a number of ticks, like for example the consequences of police investigations.

This temporal relationship is part of the information given in the activity diagram in Figure 6.1, which furthermore shows the control flow between the important behavioural elements — represented by activities — of the entire model, structured in different parts (gray background boxes). Some of the edges are labelled in order to improve readability: Temporal relations as mentioned above are put in square brackets, phrases in italics give further details on conditions if the subsequent activities do not allow to infer this information. The most important edge label is printed in bold font: the (type of) agent which is executor of the following activity. Edges with no label indicate the transition to the next activity within the same tick and as part of the behaviour of the same agent. The diagram can be read as follows.

The dynamics start with an initial normative event at the first tick regarding a random criminal. This normative event is observed by a fellow criminal at the next tick, which might adapt the image of this criminal. In case of a norm deviation event the image is changed to a lower level[84], which triggers a decision process on whether and how to perform aggressive actions against the deviating criminal. In the next tick the possibly multiple criminals which decided to sanction the norm deviation 'negotiate' and finally one of them performs a single aggression, whose consequence manifests in the next tick:

- Either the aggression is lethal, which might cause panic and 'fear for life' among other members of the criminal network, or
- the victim of the aggression survives the aggression and starts with an interpretation process.

This interpretation begins with the distinction whether the aggressor is reputable or not. In the latter case, the aggression is regarded as unjust which triggers an obligatory reaction in the next tick. If the aggressor is judged as reputable, then a normative process is performed that either leads to the conclusion that

[84]More formally, the image as an ordinal value of the variable 'image' is *decreased*. This term — together with *increase* of image for the opposite direction of change — is used in relation with the rules presented in the following.

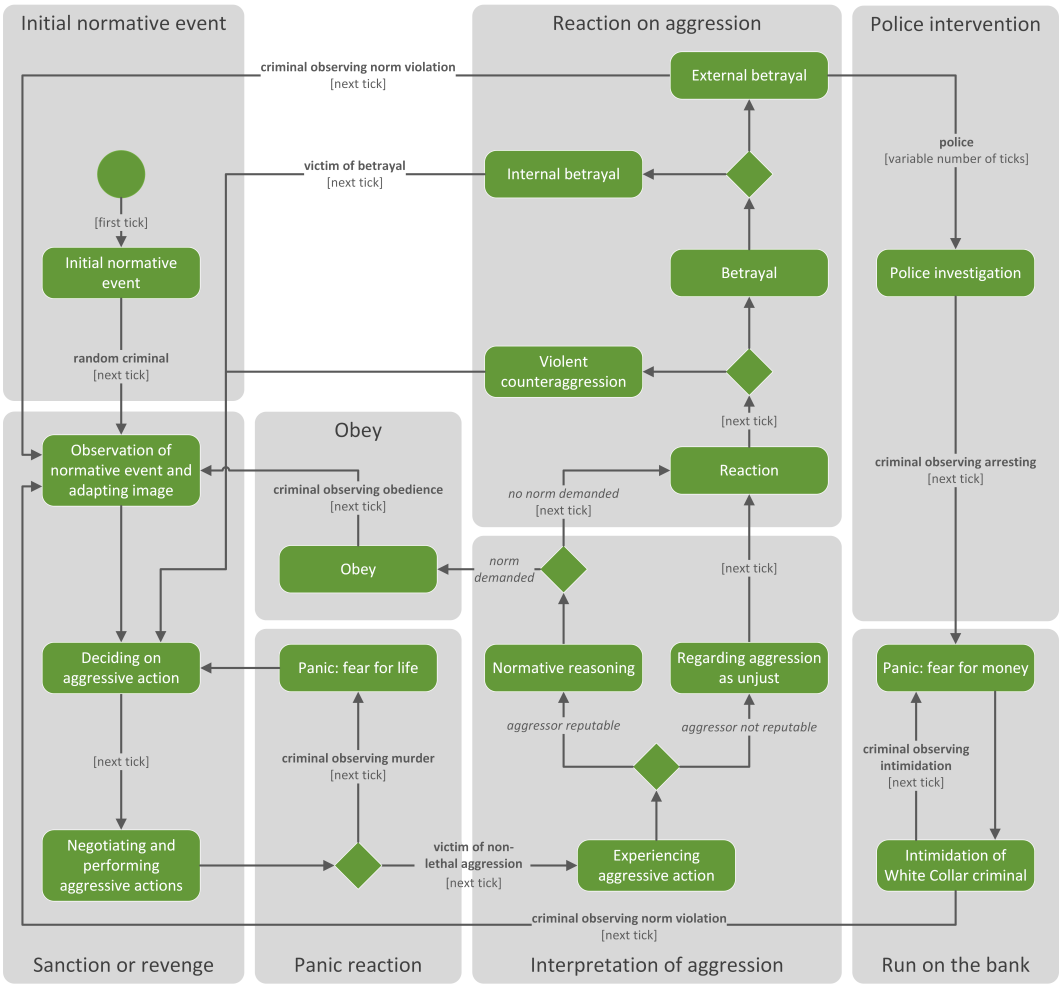

Figure 6.1: Overview UML activity diagram of simulation model (see text for meaning of edge label styles)

- a norm is indeed demanded[85], persuading the criminal to obey to the norm, which in the next tick might motivate fellow criminals to change the image of this member to a higher value, if they get to know about the obedience, or
- no norm is demanded which again triggers a reaction in the next tick.

With regard to the actual reaction a decision process is conducted (taking one more tick), with one of the following results:

- A violent counteraggression is performed, employing the same activities as for normative sanctioning (as described above), though this time executed by the reacting agent.

[85]Or 'invoked' — related to the term 'norm invocation', used for this type of event e.g. in (Conte et al., 2013).

- An internal betrayal is performed, an action targeted only against a specific victim within the network. The criminal targeted with this betrayal will decide on a responding aggression in the next tick.

- An external betrayal is performed, which can either be to inform the police, or to go to the media and revealing the criminal network (or its members) to the public. Both actions trigger police investigations, while the latter in addition is recognised as a norm violation, which might be observed by a fellow criminal in the next tick and might lead to the already known consequences of further aggressive actions.

Police investigation ultimately leads to interventions, i.e. the arresting of members of the network. This arresting might also be observed by other members and in the next tick cause a panic about the potential loss of invested money. This fear usually triggers an intimidation of the White Collar criminal, which also might be observed by other criminals, potentially starting (with a time delay of one tick) a vicious cycle of cascading acts of extortion towards the White Collar in form of a 'run on the bank'. The refusal of repaying invested money by the White Collar is at the same time regarded as a norm violation, observable by further criminals (again with a delay).

The activities shown in Figure 6.1 are described in more detail in the following sections, complementing this very brief walkthrough of the model. A number of concepts already partly introduced are repeatedly used throughout the chapter. These are:

- Rational and emotional mental frame. As mentioned above, these different 'modes of operation' of criminals influence their behaviour. In the emotional frame the criminal is less able to foresee the consequences of the performed actions, hence, the probability for severe aggressions and acts of strong violence is higher as for the rationally acting criminal (confer section 5.4.4).

- Image and reputation of criminals. Both are properties expressing the standing of a criminal, the rank in the hierarchy in a way. Reputation is initially set for each criminal agent in the initialisation of a simulation run, is known to all members of the criminal network and does not change in the course of time. In contrast, the image is an information (i.e. a fact) private to each criminal agent. I.e., each criminal has its own view on the image of each fellow criminal. The image values do change during simulation runs.

- Levels of image and reputation. These are ordinally scaled attributes: very high, high, modest, low and very low.

- Levels of severity of aggressive actions. The severity of an aggressive action is measured by the ordinally scaled attribute 'strength': low, modest, high.

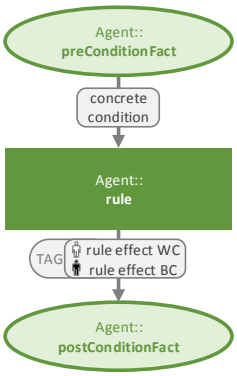

Figure 6.2: Notation for DRAMS Data-Rule Dependency Graphs

The detailed descriptions in the subsequent sections use common graphical description language, originating from the concept of the Data Dependency Graph (DDG) as introduced in section 3.3.4. Figure 6.2 gives an example of this notation. In the following, parts of the CCD action diagram discussed in the previous chapter are put in relation to the actual rule implementation, using the DDG language. It has to be noted here that the two diagram types will not only differ in the granularity of contained information (which is to be expected), but also in the reading order: the action diagram has to be read from 'bottom to top', while the DDG shows the flow of actions from 'top to bottom'.

The syntax of the DDG language is defined as follows: Ellipses represent facts. Fact names beginning with 'R_' are generated by CCD2DRAMS and are associated to relations in the CCD actor network diagram. Facts with other names correspond to objects or actors if generated, or are added to the simulation model without direct counterpart in the CCD. Rectangles represent rules, following the same notation as in the DDG automatically generated by DRAMS. There are annotations attached to edges in rectangles with rounded corners. Such annotations on edges from facts to rules specify in detail the condition under which the rule can fire. Annotations on edges from rules to facts exactly describe the information produced by the rule and stored in facts as the post-condition. Tagged annotations are highlighting candidates for simulation model parameters. The annotations can contain information differentiated for the agent types: a white actor symbol points to the White Collar criminal, black to the Black Collar criminal.

6.4 Initial normative event

To create the initial event that a member of the criminal network all of a sudden becomes disreputable — as discussed in section 5.4.2 — a global rule

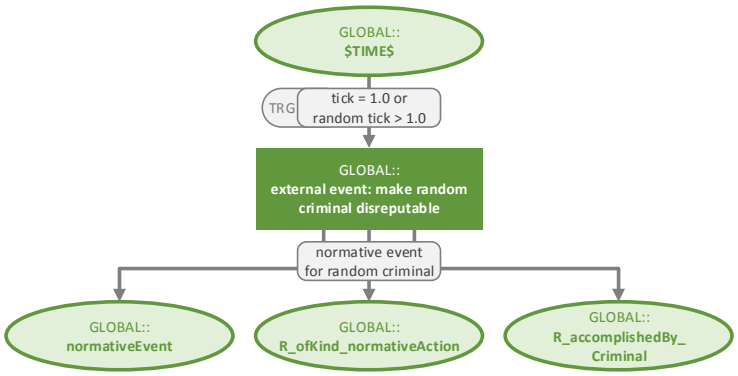

Figure 6.3: DRAMS rule: initial normative event

`external event: make random criminal disreputable` is provided (Figure 6.3). This rule does exactly what the name suggests: it randomly picks one of the members and issues a normative event about an alleged violation of the norm of trust by this member. This rule is triggered once at tick 1.0 and randomly at later ticks (TRG).

6.5 Reaction of criminal network members on normative event

The CCD action `perform aggressive actions against member X` (Figure 6.4) is an example where the derived implementation has to be way more convoluted than the action might appear at a first glance. Reason for this discrepancy in granularity is the fact that for this action not much evidence is available — the internal decision processes of criminals that lead to aggressive actions have to be regarded as a black box. Therefore, the mechanisms have to be constructed in an at least plausible — and where possible well informed — way. Key objective always is to reproduce the observed results of these decisions (for which evidence is available) from the simulations.

In Figure 6.5 the structure of the declarative rules that formalised the first part of this action is shown. The initial condition — the normative event — is comprised in the formal model by three facts, expressing the existence of a normative event of a certain kind (a normative action), accomplished by a criminal. Each fellow criminal is in principle able to perceive this event, in order to perform a reaction. Perception involves both the observation of the event, but also the 'willingness' to care about the event. This perception is implemented in the rule `react on normative event regarding fellow criminal`, which is modelled as a stochastic process:

As annotated in (A1), the White Collar criminal perceives this event with a very low probability of 0.05 since its human counterpart typically keeps out of

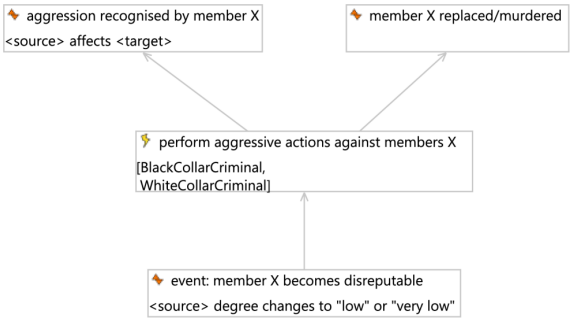

Figure 6.4: CCD action `perform aggressive actions against member X`

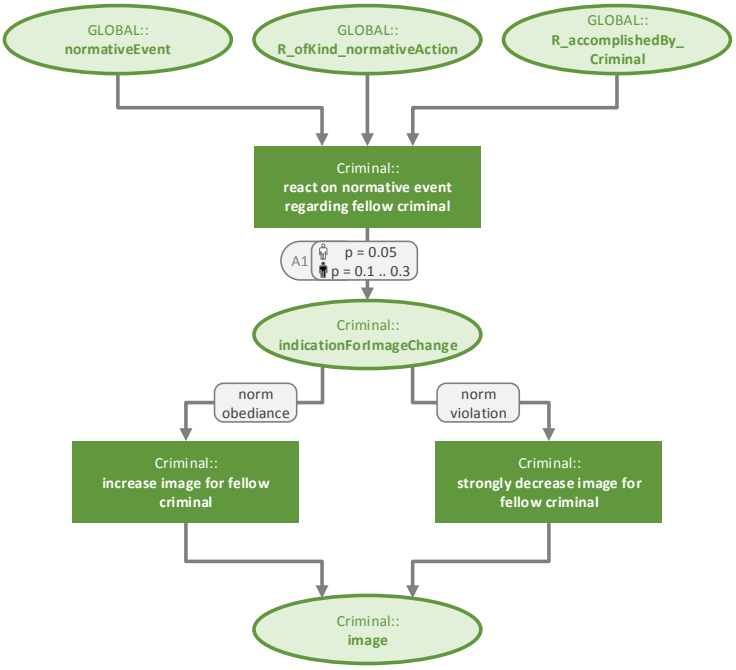

Figure 6.5: DRAMS rules: consequences of observed normative event

the thuggish business of the Black Collar criminals, while for the Black Collar criminal the probability is dependent on the image of the criminal respective to the event: with a very high image, the probability is 0.1, with high image 0.2 and otherwise 0.3. Rationale behind this differentiation is that a norm deviation of a criminal with higher image seems less likely to be an offending act against fellow criminals or a threat for the entire network.

The first step of the process that is triggered on successful perception is a change in the image of the criminal. If the normative event is a norm violation, then the image is strongly changed to a lower value ('two levels'), in the case of norm obedience the image is changed by one 'level' to a higher value.

The new image of the criminal related to the normative event then triggers the second part of the reaction (Figure 6.6), where the behaviour differs for criminals that are in the rational or emotional mental frame.

In both cases an aggression is planned only if the new image of the criminal related to the normative event is low or even very low (B1 and B2), but in the case of rational frame the planning is followed only with a probability of 0.9 (B3), whereas an emotionally acting criminal would always punish (because he might not be able to foresee the consequences of his aggressive actions; B4).

If basically the plan is conceived, then again the category of criminal (Black or White Collar) and the mental frame determine the type of reactions, but in some cases also the image of the criminal to be punished (decisions B5 and B6 of Black Collar criminal).

A rational White Collar criminal will always (B5) perform the — compared to the other options — mild punishment of (internal) betrayal, while in emotional frame he will always answer with violence (B8). A rational Black Collar criminal takes the option of betrayal only if the target of the aggression has still a low image (B5); in the case of very low image (B6), the only appropriate action is considered to be threat. An emotionally acting Black Collar criminal tends more toward a violent reaction (probability of 0.7; B8) than a threatening action (probability of 0.3; B7). The cognitive heuristics modelled in these decisions are suggested by information from the evidence base; this connection to evidence becomes more concrete when deciding on the actual aggressive action. This is implemented by four rules:

- `plan betray - rational`: one of the actions of internal betrayal in Table 6.1 is selected by chance with the specified probability (B9).
- `plan threatening action - rational` randomly selects one of the threatening actions from Table 6.2 (B10).
- `plan threatening action - emotional` same as for the rational frame (B11).
- `plan violent action - emotional` randomly selects one of the acts of violence from Table 6.3 (B12).

The finally decided aggression is then expressed 'globally' (or rather among the agents that decided to react) by the rule `perform aggressive actions against member X` (where member X — of course — stands for the criminal related to the normative event). This individual decision is composed by three facts: the individual aggression as the actual result of the decision process, information on the decided means of aggressive action, and information about the affected criminal.

Eventually, a single aggression is perpetrated against the criminal X. I.e. not all the individually decided aggressive actions are performed, but after some kind of 'negotiation' among the potential aggressors, one aggressor and

Acts of internal betrayal	**Severity**[86]	**Cases in evidence**	**Inferred Probability**
starting affair with girlfriend	low	2	0.666
stealing capital (drugs or money)	high	1	0.334
refuse to return entrusted money	high	0[87]	0.0

Table 6.1: Cases from evidence informing the probability for internal betrayal ([86]in the simulation model code severity is called 'strength'; [87]Special type of internal betrayal which becomes relevant only in the case of extortion of the White Collar criminal, which according to the evidence occurred with quite a high frequency. Dedicated rules (see further below) take this into account.)

Acts of threat	**Severity**	**Cases in evidence**	**Inferred Probability**
having an eye on you	low	9	0.264
outburst of rage	low	6	0.176
demanding money in dark forest at midnight	modest	4	0.118
death threat	high	11	0.324
kidnapping	high	3	0.088
putting a gun into the stomach	high	1	0.030

Table 6.2: Cases from evidence informing the probability for threatening actions

Acts of violence	**Severity**	**Cases in evidence**	**Inferred Probability**
beating up	modest	3	0.143
attempting to kill	high	18	0.857

Table 6.3: Cases from evidence informing the probability for violent actions

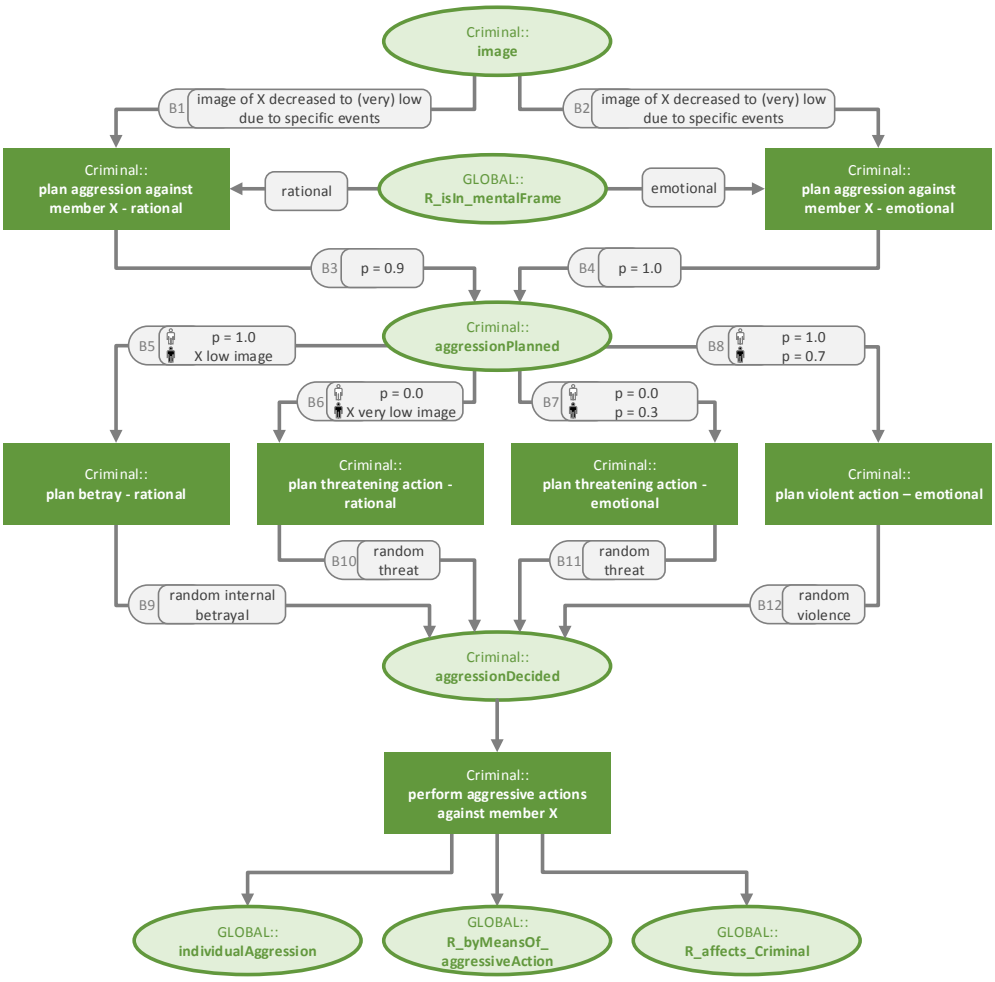

Figure 6.6: DRAMS rules: reaction on change to low or very low image of fellow criminal

the related aggression are selected. This process as shown in Figure 6.7 is not associated to any agent, but part of the 'global' environment. The selection process is two-staged. In the first stage (rule `find candidates among individual aggressors`) the criminal with the highest image is chosen (C1). In the second stage (implemented in rule `coordinate aggression among individual aggressors`) the aggression with the highest severity (as referred to in Table 6.1 to Table 6.3) is the selected, potentially involving a stochastic process if more than one candidate fulfils these criteria (C2).

The subsequent (implicit) execution of the aggression is immediately evaluated in terms of impact for the victim. For acts of violence a certain (quite low) probability for lethal consequences is considered (C3; 0.2 for murder attempt, 0.1 for beating-up). All other possible types of aggression are not assumed to be lethal, anyway.

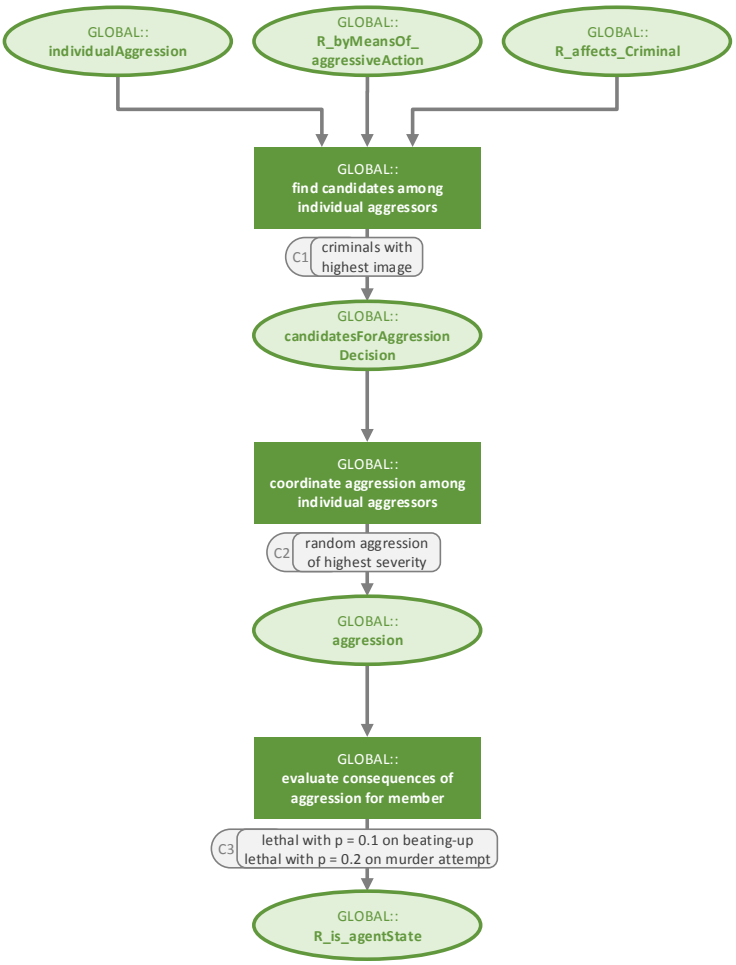

Figure 6.7: Global DRAMS rules: negotiation and execution process of a single aggressive action, selected among the potential aggressores

6.6 Reasoning and reacting on aggressive action

The action in the CCD action diagram following the performing of the aggressive action is the interpretation by the victim (Figure 6.8). As shown in the diagram snippet, in addition to the condition focussed here — the aggression motivated by an alleged norm deviation "recognised by member X" — there are two other circumstances when a criminal becomes victim of an aggression: either as a result of a counteraggression or — as a special case — the intimidation of the White Collar criminal. Both cases are not directly linked to a normative event, and become relevant at later stages in the dynamics. This interpretation process remains the same for all cases. In the implementation it is assumed that the victim always perceives the aggression against itself. In

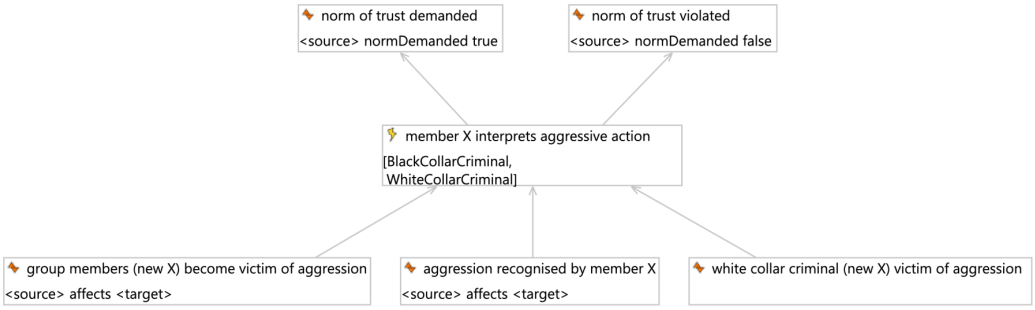

Figure 6.8: CCD action `member X interprets aggressive action`

the following, the implementation of the quite complicated reasoning process is spread out in detail.

This interpretation process, again, consists of several sub-processes. The first, as shown in Figure 6.9, covers the perception of the aggression and collection of relevant data related to the aggression (e.g. about the attacker and the actual kind of aggression; rule `member X interprets aggressive action`), followed by a first evaluation of the appearance of the attacker — deduced from the attacker's reputation (rule `is attacker reputable`). As indicated by (D1), the information whether the attacker is reputable (high or very high reputation) or not (modest or lower reputation) is memorised in a respective fact.

Depending on this reputation information, the interpretation is fundamentally different. For the case of a reputable attacker the process is depicted in Figure 6.10. The first rule is — as the 'normative process' — the heart of this branch of behaviour. The normative process encapsulates the reasoning about whether the attacked criminal might have violated a norm in the recent past which would have led to a sanction by another fellow criminal. The basic idea of this normative reasoning is quite simple: It is evaluated whether aggressive actions performed in the past by a criminal stand in some kind of temporal relationship with a normative event assigned to this criminal. In order to conduct this evaluation, each criminal can access a global event board[88] where all aggressions performed by each criminal are recorded. Also the normative events are logged in a similar way, so that temporal relations between these types of events can easily be derived. The normative process is considered successful, if aggressions are found which at most 16 ticks later led to normative events (E1). If such relations exist, the criminal regards a norm as demanded (or 'invoked') and memorises this result in a fact for norm evaluation, in order to react accordingly later on.

[88]This global event board can be seen as a kind of collective memory, fed e.g. by gossip-like spread of information.

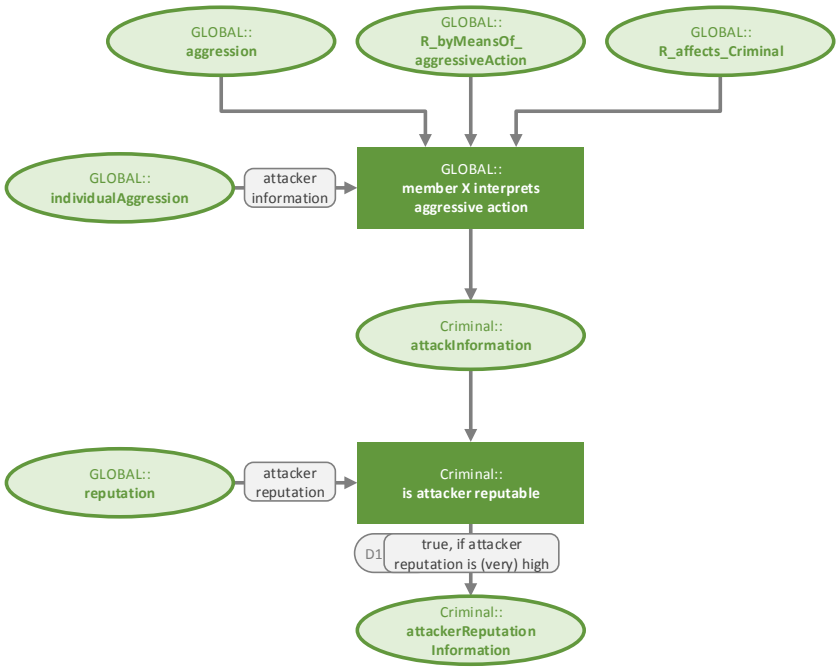

Figure 6.9: DRAMS rules: experiencing aggressive action

Even if the normative process failed, the aggression might still be regarded as a justified sanction: If either the attacker has a high or very high image, or the aggression was mild or modest (E2), then it is assumed that a norm is demanded as well. This cognitive heuristic has been included in the model to cover the possible aptitude of criminals with high image (and high reputation) to mitigate conflicts, either by mediating or by just exercising authority.

In contrast, if either the attacker's image is modest or low, or the aggression was of high severity ('strong aggression'; E3), then the aggression is perceived as arbitrary, which means that no norm can be demanded. As victim of such a kind of aggression, the change into the emotional mental frame appears to be indicated.

For the case of a non-reputable attacker the process is much simpler (Figure 6.11). As results of the rule `impact of attack by not reputable attacker` this fact is firstly memorised (to trigger a counter-aggression later on), but at the same time the mental frame potentially might change to emotional (due to fear or rage), namely in case of strong or modest violence or strong threat (F1). The actual switching into the emotional frame is performed by the rule `switch to mental frame`.

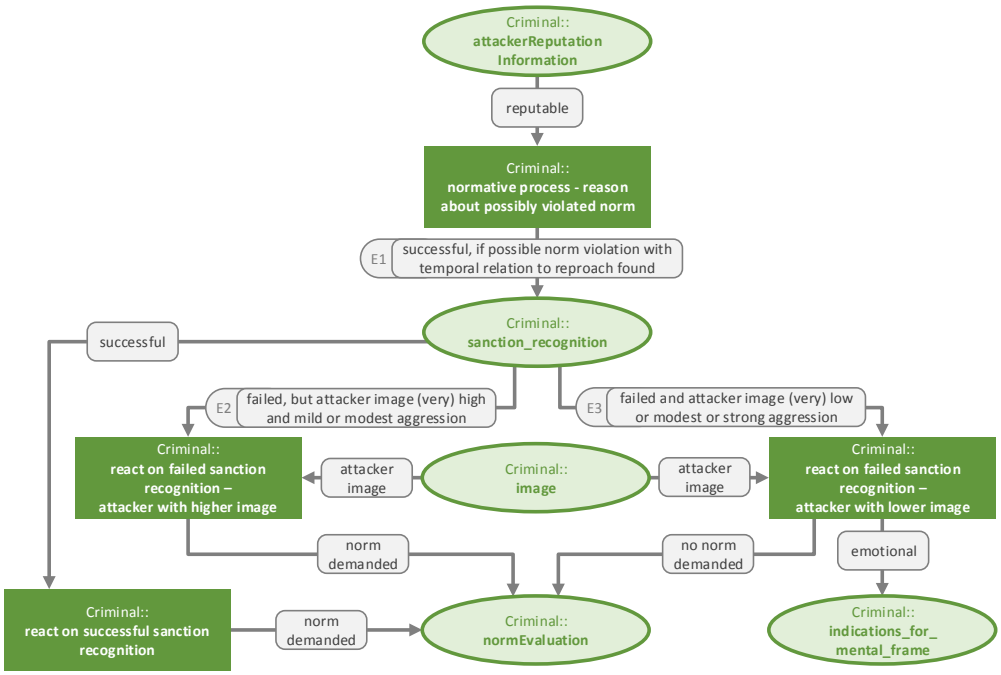

Figure 6.10: DRAMS rules: normative reasoning, if aggressor is reputable

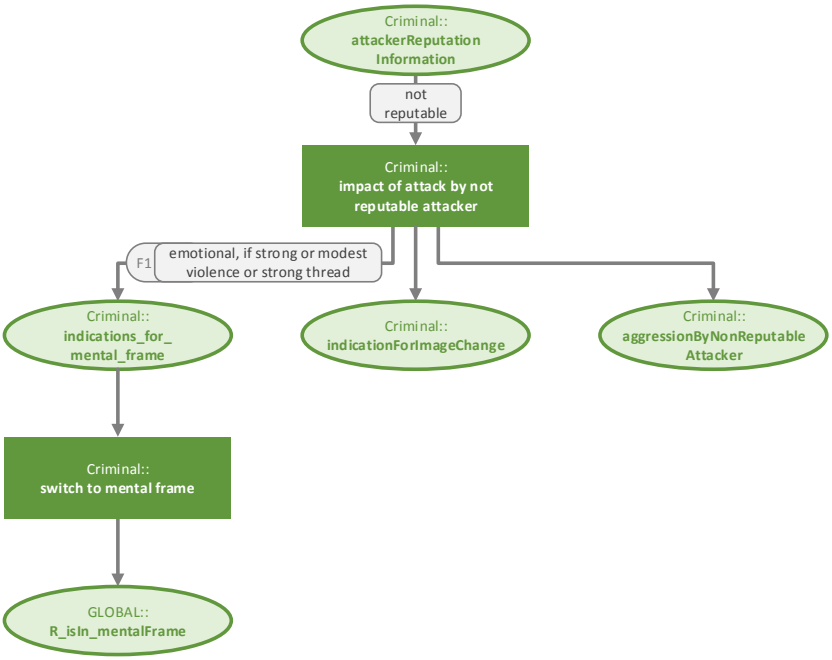

Figure 6.11: DRAMS rules: impact of attack by non-reputable aggressor

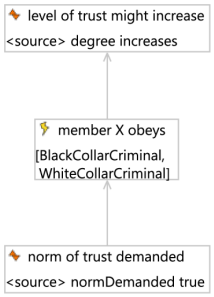

Figure 6.12: CCD action `member X obeys`

6.6.1 Norm Demanded: Obey

The action diagram fragment shown in Figure 6.12 covers the criminal's reaction if the normative reasoning resulted in the insight that the experienced aggression is likely to be justified, be it as a sanction for an own actual norm violation or just due to the high image of the aggressor. The only possible action implemented here is to obey the aggression in order to recover the trust among the criminals.

The respective simulation model presented in Figure 6.13 foresees two rules for this obeying behaviour, one for each of the two circumstances to obey as mentioned above:

- `member X obeys`, if the normative process classified the aggression as a sanction, and
- `member X obeys due to high image of aggressor`.

The result in both cases is the same: a normative event carrying the message that the criminal is willing to obey to the norm of trust is sent to the environment, i.e. made known to the other members of the criminal network.

This requested normative event is enacted by the global rule `activate normative event` (Figure 6.14). Purpose of this rule is to randomly select among possibly several normative events regarding each criminal (G1). E.g. for a criminal that has just obeyed, at the same time another normative event about another norm violation might occur. In this case only one normative event is activated with the consequence that if a norm reproach is issued, then a possible act of obeying (to another norm reproach or a sanction due to high image) at the same time might not become known, or the other way around. Hence, this rule plays an important role also in different scenarios, when e.g. a norm violation event takes place (see below).

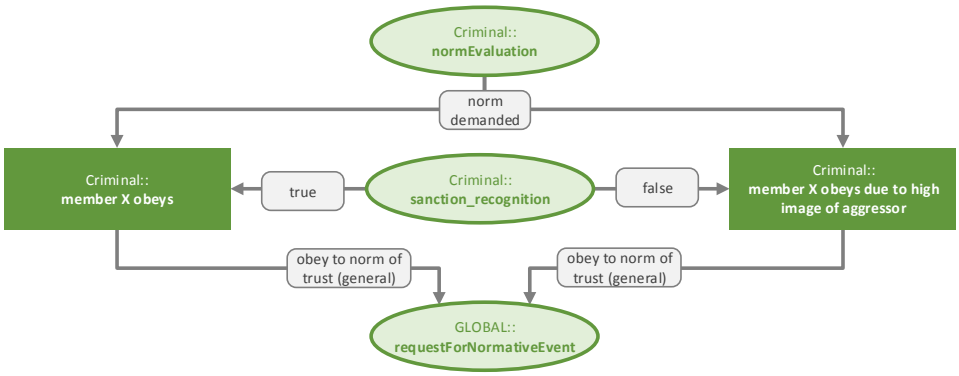

Figure 6.13: DRAMS rules: variants of norm obedience due to perceived sanction

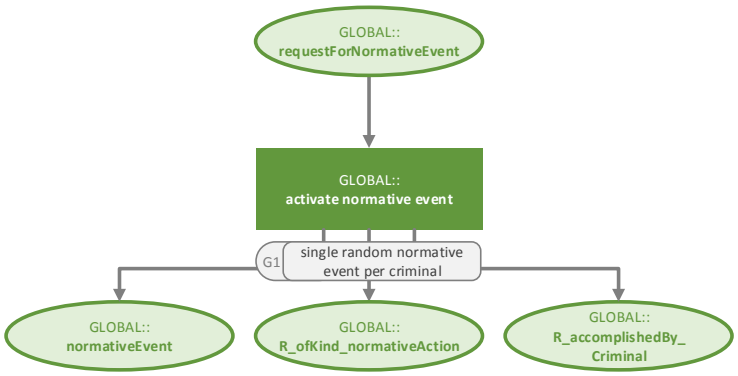

Figure 6.14: Global DRAMS rule: public enacting of normative event

6.6.2 No Norm Demanded

The opposite result of the normative reasoning is the awareness that the aggression cannot be a justified sanction or the aggressor has such a low image that it is ineligible to be a sanctioner (i.e. aggression by non-reputable attacker). This particular instance only leaves margin for two types of reaction, either betrayal or violent aggression. This pre-selection process as shown in Figure 6.15 has no counterpart in the CCD, but is implicitly modelled in the different actions branching from the condition `norm of trust violated` between Figure 6.8 and the action diagram fragments in the following two subsections. The rule `decide on adequate reaction` is shown twice, as it behaves differently if the criminal is in rational or in emotional frame. For the former case, a reaction is decided with a probability of 0.4 to be betrayal and 0.6 to be (violent) aggression (H1). For the latter case the probabilities for the two options are just the opposite (H2).

Although the rules in the simulation model part related to 'reaction on

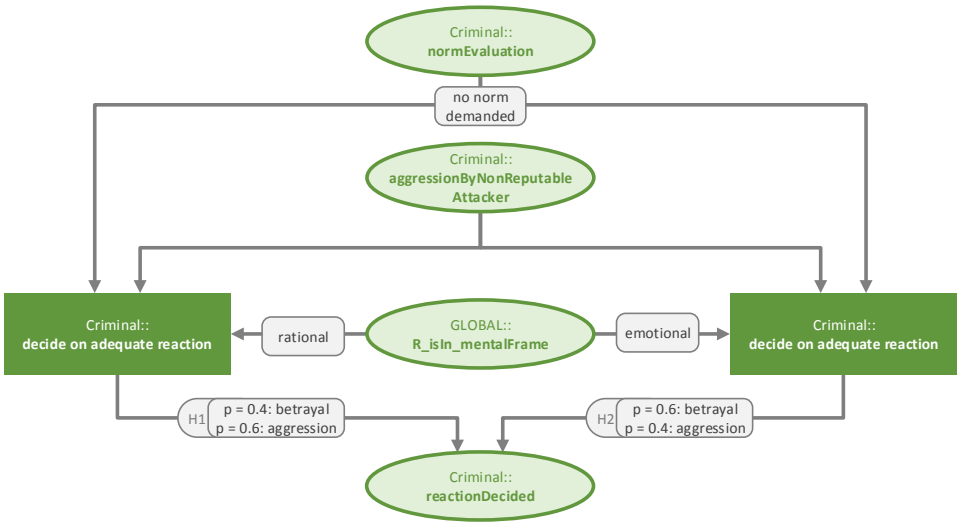

Figure 6.15: DRAMS rules: deciding on violent or treacherous reaction, if aggression was not perceived as sanction

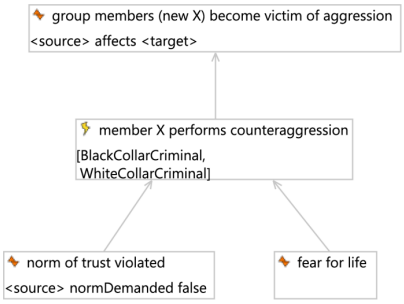

Figure 6.16: CCD action `member X performs counteraggression`

aggression' are quite similar in their functionality to the rules for deciding on a reaction on decreasing image (e.g. due to a norm violation, as described above), the actual implementation is different. Reason is that here the implementation follows the CCD action diagram much closer, as more concrete evidence is available for these parts of the model.

6.6.3 Counteraggression

One of the two options to respond to unjust aggression (and also to a panicky fear for life, as described later) is to perform counter-aggression, which in this context means some kind of violent act. The action diagram part in Figure 6.16 translates into the DRAMS implementation specified in Figure 6.17.

The initial rule `member X performs counteraggression` of this decision

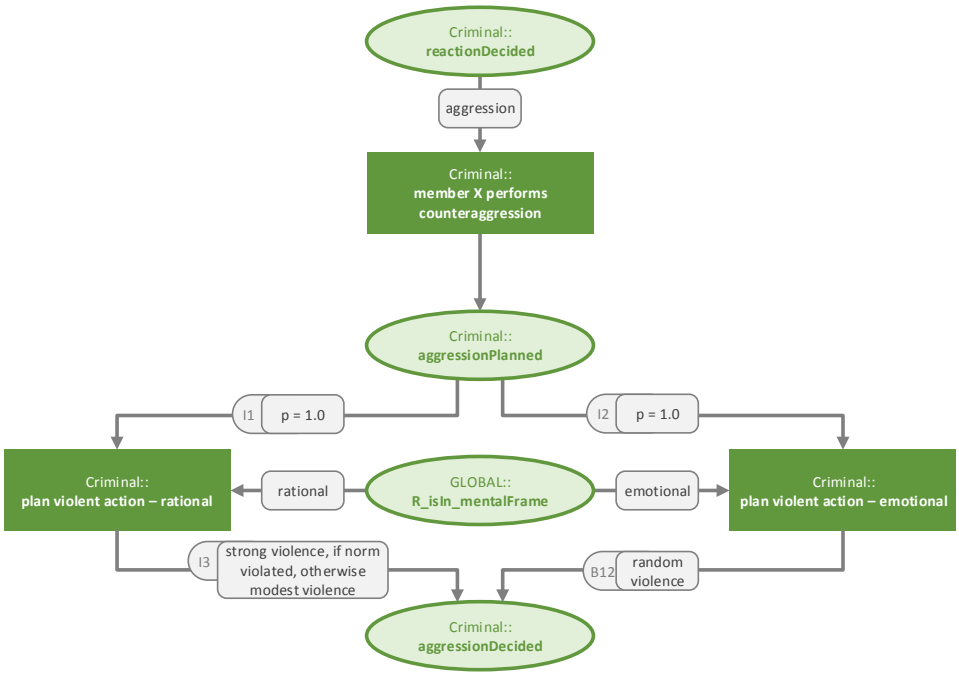

Figure 6.17: DRAMS rules: deciding on type of violent action as counteraggression

process just decides whether to actually perform a counteraggression. In the implementation it is adjusted in a deterministic way, as for both situations — that the criminal is in rational mental frame (I1) or in emotional mental frame (I2) — the probability is 1.0 that one of the rules for planning a violent action will fire.[89] The rule for the emotional frame (`plan violent action - emotional`) is the same as in Figure 6.6 and already described in the related section. Therefore the result of the rule refers to annotation (B12).

The rule for the rational frame (`plan violent action - rational`) works slightly different (I3): If the target of the aggression (the original aggressor) was accused of a norm violation before, only strong violence (i.e. of high severity) is applicable. Otherwise, a violent action with modest severity is to be performed. For possible violent actions and associated probabilities for both rules, confer Table 6.3.

6.6.4 Betrayal

The second option to respond to the conditions listed in the previous section and shown in Figure 6.18 is to betray the criminal network. There are basically two possible categories of betrayal: the quite harmless internal betrayal,

[89]Meaning that this rule is a 'dummy' in the current implementation, with the possibility to be changed in the future, if respective evidence would demand this.

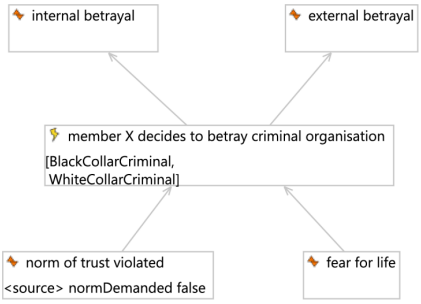

Figure 6.18: CCD action `member X decides to betray criminal organisation`

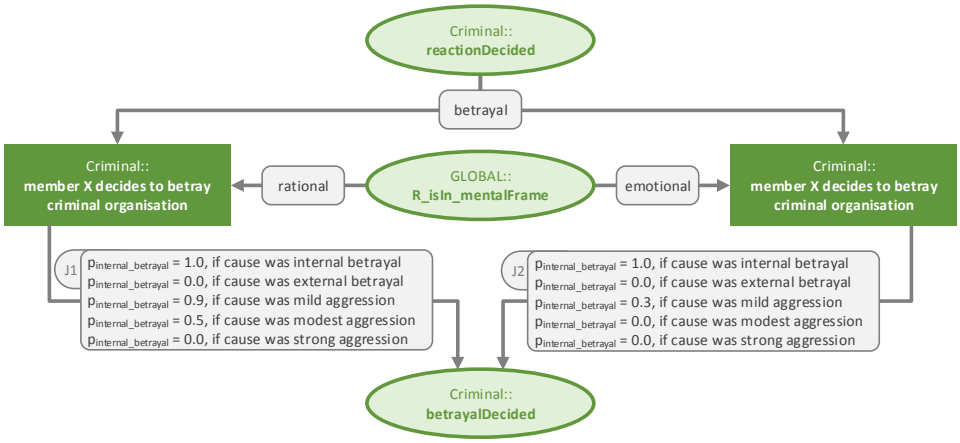

Figure 6.19: DRAMS rules: deciding on type of betrayal — internal or external

and the serious and (for the criminal network and the individual) existence-threatening external betrayal.

The decision about the type of betrayal is implemented in the rule `member X decides to betray criminal organisation` (Figure 6.19). The effect of the rule is again influenced by the mental frame of the deciding criminal. The probabilities for internal betrayal under certain conditions are detailed in (J1) for the rational frame and in (J2) for the emotional frame. The probabilities for external betrayal are then $p_{external\_betrayal} = 1 - p_{internal\_betrayal}$ for each case. As a short summary, if the original aggressive action was internal betrayal, then the reaction will always be internal betrayal, too. If it was external betrayal or any other kind of strong aggression (threat or violence), then internal betrayal as a reaction is never an option. In all other cases there is a probability ≥ 0 for internal betrayal.

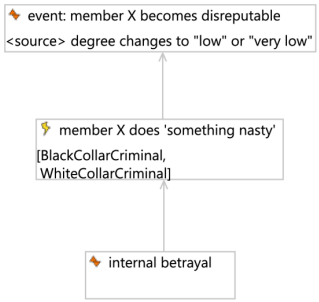

Figure 6.20: CCD action `member X does 'something nasty'` (internal betrayal)

Internal Betrayal

If the choice in the previous stage of the decision process is internal betrayal, then a 'nasty' action invisible for the environment outside the criminal network is performed (Figure 6.20).The consequence for the attacker to be expected is to become disreputable in a similar way as it is the case with the initial normative event, provoking respective aggressive actions.

The diagram in Figure 6.21 not only shows this performing of the internal betrayal by the attacker (rule `member X does 'something nasty'`), but also the reaction of the betrayed 'target' criminal. The internal betrayal is always a covert aggression against the target, and the act of betrayal is selected randomly (K1) among the possible options and related probabilities listed in Table 6.1.

The target criminal detects this covert aggression as internal betrayal (K2) and decides on how strong this action leads to a decline of image of the attacker, implemented in the rule "react on covert aggression". If the act of internal betrayal is of mild or modest severity, then the image is slightly decreased by one level (K3), otherwise strongly by two levels (K4). This new decreased image might set off aggressive behaviour as formulated in Figure 6.6.

External Betrayal

The two options for external betrayal are shown in Figure 6.22: Either the criminal provides hints (or a criminal complaint) directly to the police, or details of the criminal network or associated activities are revealed to the public (and, hence, also to the police) by informing newspapers or other media.

The implementation of the decision process on external betrayal and the performing of the actual acts of betrayal are depicted in Figure 6.23. The rule `decide type of external betrayal` randomly selects between the options with certain probabilities as taken from the evidence base (L1), see Table 6.4.

If the decision is to inform the public, the rule `member X goes to public` asserts a globally visible fact about the existence of the criminal network

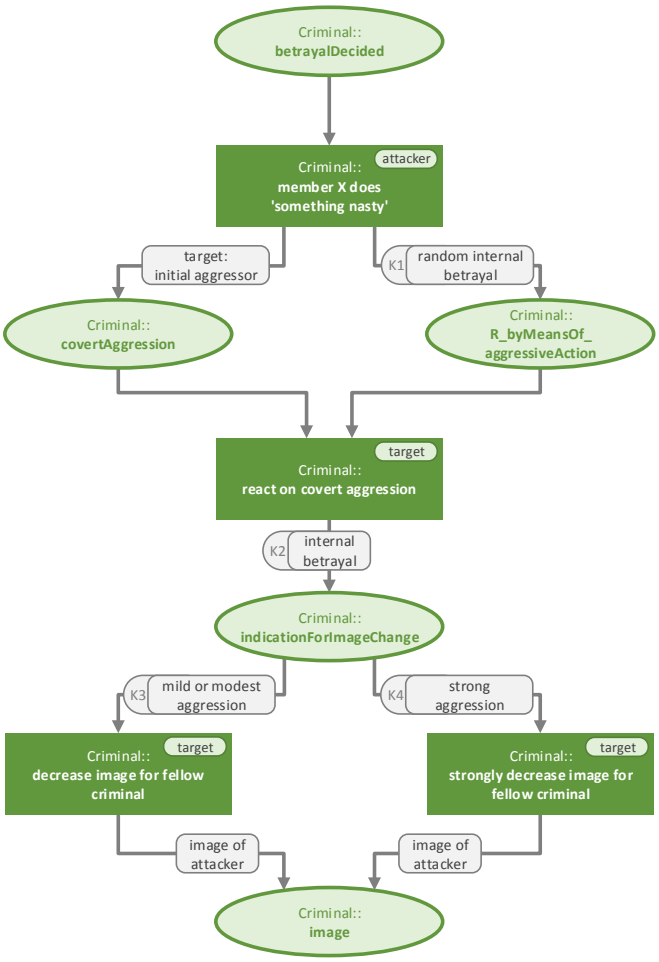

Figure 6.21: DRAMS rules: process and consequences of internal betrayal

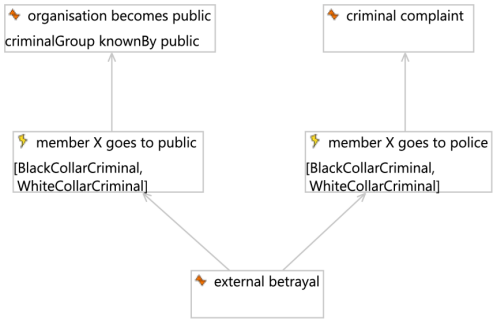

Figure 6.22: CCD actions for external betrayal: `member X goes to public` or `member X goes to police`

(`R_knownBy_public`), but also puts this action on the criminal's 'tally stick'

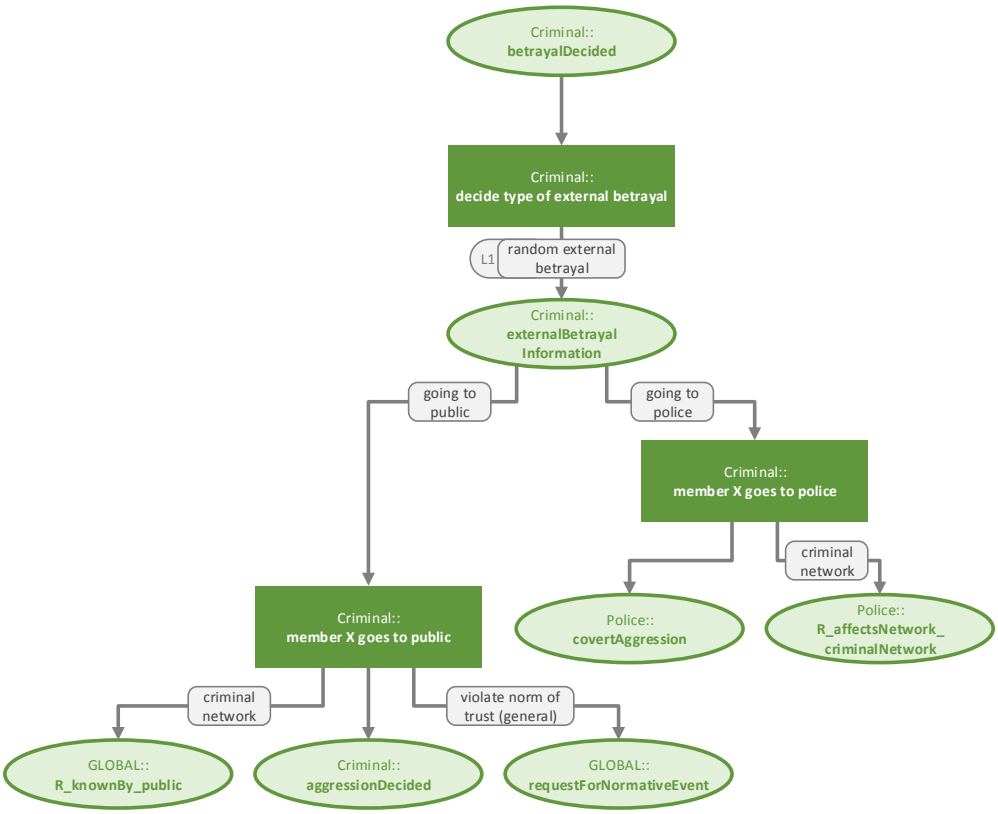

Figure 6.23: DRAMS rules: deciding on type and performing of external betrayal

(fact `aggressionDecided`) and also sets the grounds for having this activity be regarded as a violation of the norm of trust (global fact `requestForNormativeEvent`).

The consequences of informing the police are different, as this is a covert aggression (fact `covertAggression`), involving the betraying criminal and the police only, and invisible to the other criminals. As a result, the police now knows about the existence of the criminal network (fact `R_affectsNetwork_criminalNetwork`).

6.7 Panic

The central cognitive element among the individual agents of the criminal network in terms of escalating aggression and violence is the appearance of panic, i.e. a situation where rational deliberations do no longer play a role in the behaviour of the individual criminal. Here two kinds of panic are distinguished: fear for life and fear for money. While the former is mainly triggered

Acts of external betrayal	Severity	Cases in evidence	Inferred Probability
going to police	modest	2	0.334
going to public	high	4	0.666

Table 6.4: Cases from evidence informing the probability for external betrayal

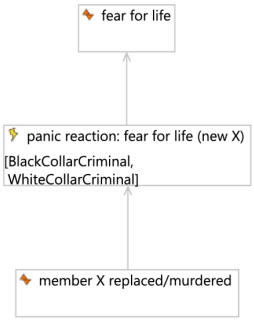

Figure 6.24: CCD action `panic reaction: fear for life (new X)`

by obtaining knowledge about a murder of a fellow criminal, the latter can have different causes where the repayment of black money handed over to the White Collar criminal might be in danger. The following subsections provide more details of these two panic reactions.

6.7.1 Fear for Life

The CCD action for `fear for life` panic in Figure 6.24 shows only one possible pre-condition: another member of the criminal network got murdered[90]. Although this is a simplified implementation, it covers the aspect of loss of trust in the network quite well, based on the loss of image of individual fellow criminals.

Thus, the implementation in Figure 6.25 becomes quite simple. As soon as a murder of a fellow criminal is observed the 'fear for life' panic state is established with a probability of 0.5 (M1) by the rule `panic reaction: fear for life (new X)`. The criminal switches into emotional frame, and becomes active in some way in order to 'defend' itself. Here the rule `action caused by fear for life` just picks randomly one of the fellow criminals (M2) — which is believed to be the one guilty for the murder as perceived by the criminal in panic — and just sets the related image to a lower level. This decrease of image might then trigger further actions, as implemented in Figure 6.6. Hence, the spiral of violence can escalate.

[90]This is a simplification of the original action diagram provided as part of the model repository (see appendix B), where also a more general panic might lead to fear for life, if

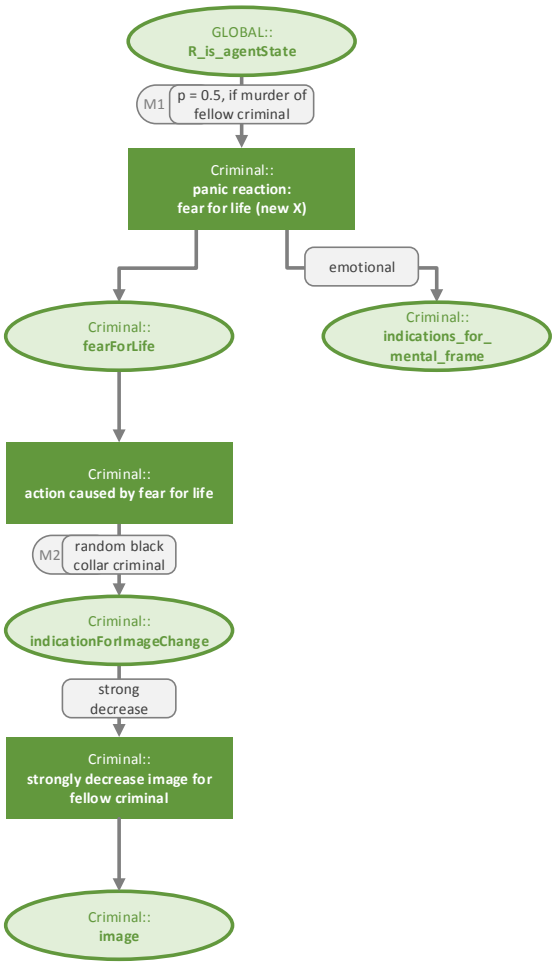

Figure 6.25: DRAMS rules: process and consequences of panic due to fear for life

6.7.2 Fear for Money

The other kind of panic — the fear to lose invested money — is modelled in the CCD as the action shown in Figure 6.26. Three possible pre-conditions for this panic are envisaged: the arresting of a member of the network, the knowledge about intimidating activities against the White Collar criminal and the circumstance that the network is known to the public. In the implementation a slight variation is realised as the latter of these three conditions is not taken into account. This simplification is justifiable because (as described later in the chapter) the uncovered network triggers police investigations that ultimately lead to arresting of criminals, which then causes panic reactions anyway.

The entering of the panic mode is implemented by the rule `panic reac-`

the overall trust in the criminal network has been destroyed.

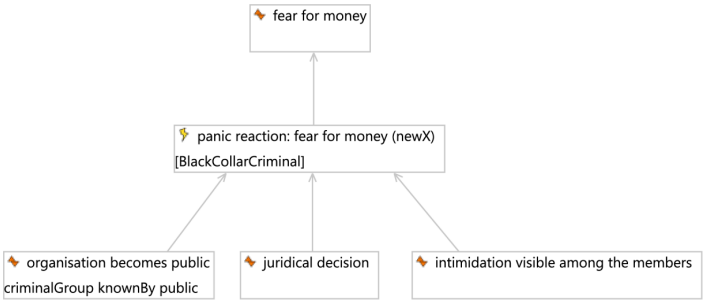

Figure 6.26: CCD action `panic reaction: fear for money (new X)`

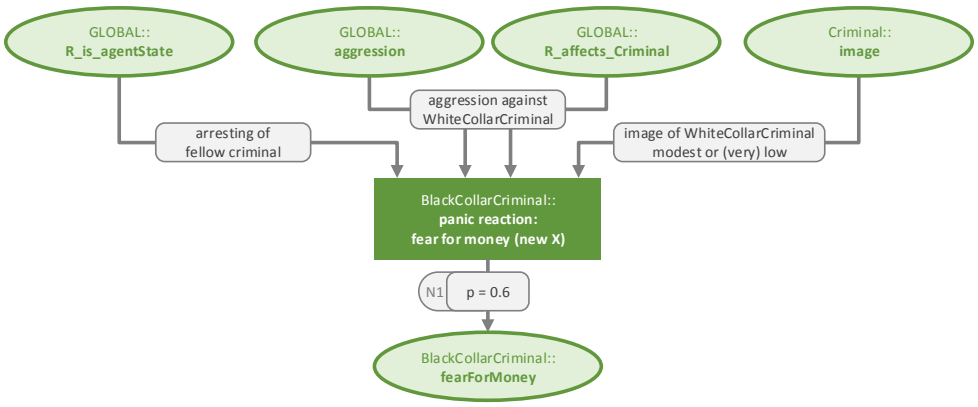

Figure 6.27: DRAMS rules: causes for panic about loss of money

`tion: fear for money (new X)` (Figure 6.27). The panic sets in with a probability of 0.6 (N1) if one of the following conditions holds:

- the state of a fellow criminal changes to 'arrested',
- an aggression against the White Collar criminal is observed, or
- the image of the White Collar criminal decreases to a modest or worse level. This case is not explicitly modelled in the CCD, but becomes important for the dynamics when the White Collar is involved in a conflict, and the opposite party of the conflict at some time responds with requesting the invested money back.

The consequences of this panic mode are described in detail in the subsequent section, as it is — according to evidence documents — related to one of the most prominent violence dynamics observed in the criminal network: the 'run on the bank'.

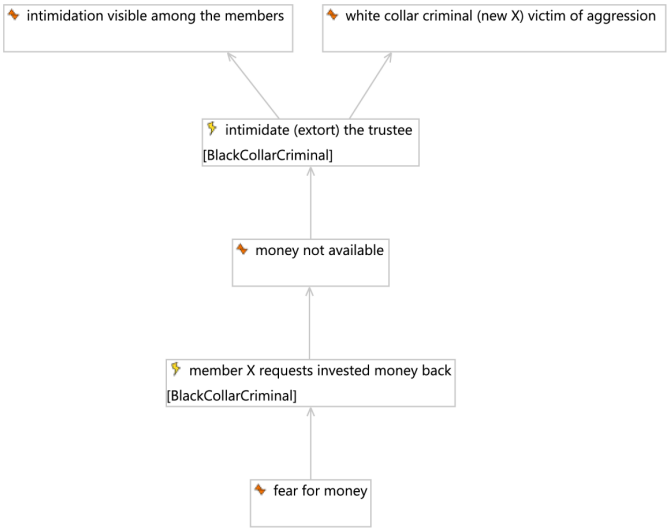

Figure 6.28: CCD actions for intimidating the trustee (WhiteCollarCriminal): `member X requests invested money back`, followed by `intimidate (extort) the trustee`

6.8 Intimidation of criminal network members

The intimidation of the White Collar criminal — in this context called the 'trustee' — is conceptually modelled as a two-stage process, as shown in Figure 6.28. When a (Black Collar) criminal gets into a panic because of fear to lose the invested money, an approach to get the money back is started. If the White Collar is unable to return the money, the actual intimidation — often in shape of an extortion attempt — takes place. This extortion results on the one hand in aggressive actions against the trustee, but on the other hand might be observed or become known by other members of the criminal network. The latter leads to the escalating number of approaches to get hold of invested capital, a typical panicky situation often referred to as 'run on the bank'.

The mechanism with which this behaviour part is implemented is deterministic. In Figure 6.29, the fear for money triggers the rule `member X requests invested money back`, which is expressed by four facts:

- `crime` — the actual request. The name 'crime' is used in this context because this act of requesting illegal (laundered) money can be considered as a crime from the perspective of the legal world.
- `R_involves_Criminal` — the White Collar criminal from which the money is requested.
- `R_performedBy_Criminal` — the requesting Black Collar criminal.

- `R_byMeansOf_criminalAction` — the amount of requested money. The amount is randomly chosen between two different values (O1), a low value of 6 million units and a high value of 20 million units.

The approached White Collar criminal processes this request by trying to fulfil as many as possible of the requests arriving at the same tick (O2). A capital stock of 20 million units is available initially, which is refilled by another 20 million units each tick after some amount was requested and paid back. The decision to pay or not to pay is communicated to the requesting criminal using the same kinds of facts as for the request:

- `crime` — the response to the request for money.
- `R_involves_Criminal` — the Black Collar criminal targeted with the response.
- `R_performedBy_Criminal` — the responding White Collar criminal.
- `R_byMeansOf_criminalAction` — the information about acceptance or refusal to repay the money.

For Black Collar criminals that got their money back the crisis is resolved for the moment, and no reaction is to be expected. Hence, this case is not represented in Figure 6.29. In the other case, the refusal to return the money is formulated as an aggressive action (White Collar rule `regard non-payment as aggressive action`) that is formalised as an act of internal betrayal, namely the 'refuse to return entrusted money' (see Table 6.1). The Black Collar criminal interprets this aggression as a norm deviation. Hence, the rule `intimidate (extort) the trustee` issues a request for a normative event containing the message that the norm of trust is violated by the White Collar criminal. This normative event triggers the mechanism of revenge or sanctioning all over again.

6.9 Police Investigation

The third actor besides the Black and White Collar criminals included in the simulation model is the police as an institutional agent. The only action modelled in the CCD action diagram is the start of an investigation because of a criminal complaint or media reports, that finally results in a juridical decision, that is the arrest of a criminal (Figure 6.30).

The implementation of this action involves a few additional aspects, as Figure 6.31 unveils. Both mentioned pre-conditions trigger the rule `start investigation`, which generates an investigation report (expressed as a fact). This report contains information about the reason for and the subject of investigation, as well as the source of information. Reason can be either media report or criminal complaint, subject is the criminal network. If the subject

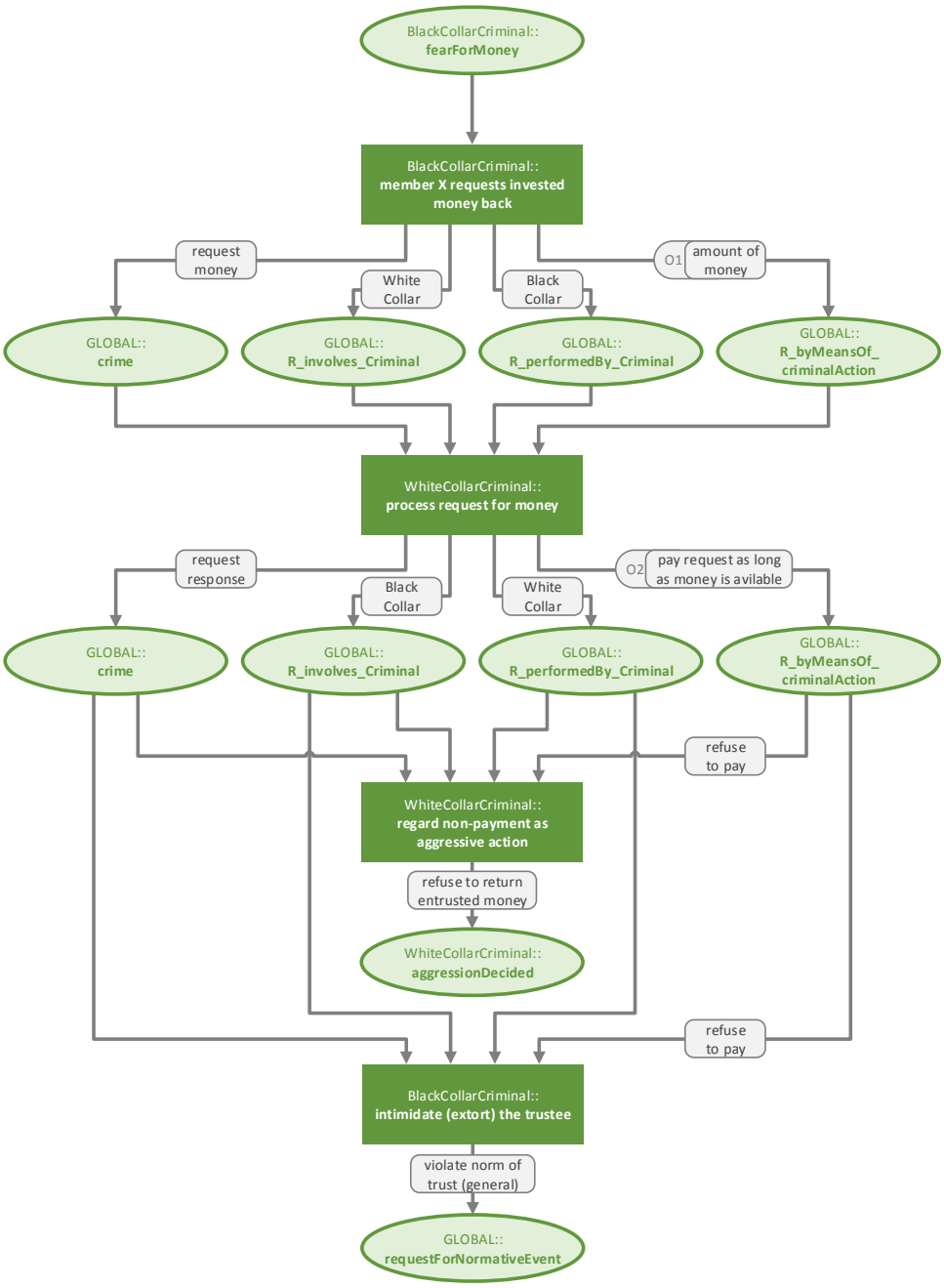

Figure 6.29: DRAMS rules: complete process of intimidation of the WhiteCollarCriminal

is unknown to the police so far, then an investigation is initiated (respective rule) and the investigation progress is set to 0 per cent. If the subject is known already, then the progress advances (with an additive calculation) by 50 per cent (P2), implemented by the rule `advance investigation`. In any case, the progress of investigation changes randomly with every tick (rule `progress`

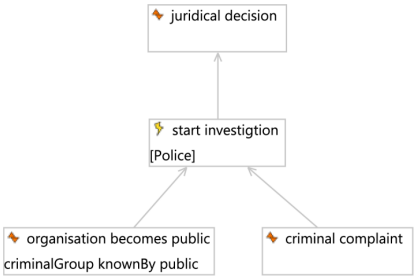

Figure 6.30: CCD action `start investigation` by police

`investigation`). This progress expectedly increases by up to 40 per cent per tick, but might also decrease by up to 10 percent per tick (P1). This negative progress expresses a possible 'dead end' in which an investigation branch might enter. If finally 100 per cent progress is reached, measures can be decided (respective rule) against the now officially identified criminal network members. A concrete measure is then taken with arresting a randomly selected criminal (P3). An arrested criminal does no longer take part in any business of the criminal network.

6.10 Conclusions on the simulation model description

The previous sections are intended to present the simulation model in a way to enable interested readers to comprehend the formalisations done on base of the conceptual model as well as the simulation experiments elaborated in the following sections, even with the possibility for model replication. However, all the technical details that are inevitable for runnable software systems can obviously not presented in the frame of a book chapter. These are aspects like the configuration of parameter settings, the control of simulation runs and the generating and visualisation of simulation outcomes. In the following a few remarks are given to each of these aspects. Simulation parameters are implemented in three different ways:

- Parameters interesting for experimentation and typically without relation to the evidence base can be put on the Repast user interface. This is done for the number and relation of reputable and normal Black Collar criminals.

- Parameters that have a close relation to the evidence base are typically modelled in the conceptual model and annotated with phrases from the evidence. Hence, these parameters have to be changed in the CCD, and a following code transformation updates the parameters in the simulation

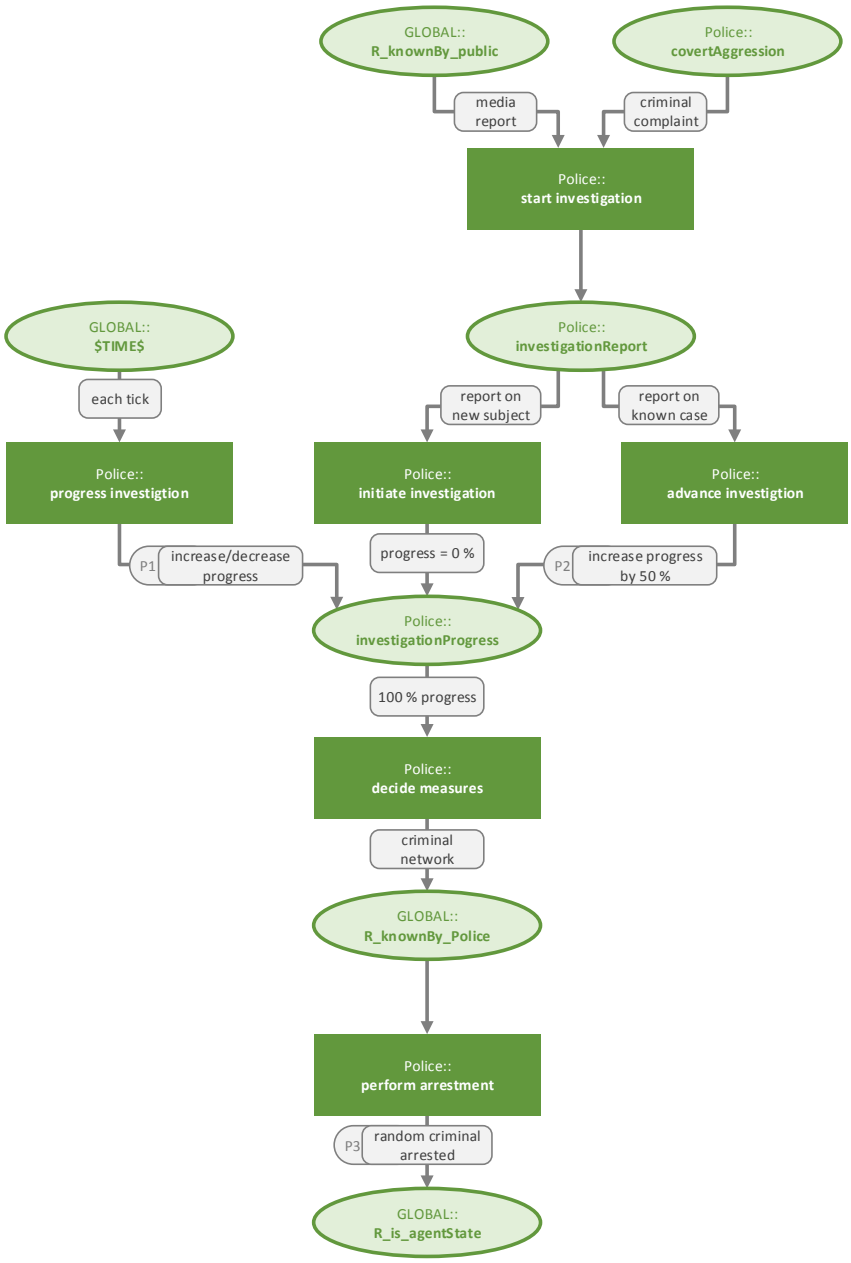

Figure 6.31: DRAMS rules for police activities

model. E.g. the types and probabilities of the aggressive actions are modelled in this way.

- All other parameters are coded in the rules, in most cases as probabilities with comments in the code.

To run a simulation, the procedure typical for RepastJ 3.1 simulation models has to be followed. Since DRAMS is just a software framework used from

within the Repast/Java code it is basically transparent to the user. During simulation runs outputs are generated which are presented and stored in different ways. The DRAMS rules produce text statements, written to a console window and stored in a log file. Per run there is also a sequence diagram generated, showing all the interactions that appeared in the run. Finally, a graphical visualisation of the agents with animations of the events happening in each tick is presented while running a simulation. The following Chapter 7 is dedicated to these topics, in relation with adequate simulation result presentation.

As an outlook, during experimentation with the model new research questions appeared, among others regarding the observed emergence of a normative authority, and how this could be established in reputation changes over the course of simulation runs, e.g. by adaptation of reputation values based on image changes. For example, if all but one agents change their images of the remaining agent, it would seem consequent to change the reputation of the latter. However, in discussions between the Data Analyst and the Simulation Modeller, a concrete mechanism could not be decided, so the implementation if this feature was postponed and the topic added to the list of items requiring further research or evidence.

Chapter 7

Generating virtual experience

7.1 Introduction

This chapter concludes the first practical part of this thesis by elaborating on the presentation of simulation results generated by the model developed in the course of the previous two chapters — with the background of model verification and validation (as defined in sections 2.2.7 and 2.3.4). Thereby, both the methodological perspective is covered, as well as the actual results created during simulation experimentation. Although similar and related material has been published before (e.g. by Lotzmann and Neumann (2016)), the content of this chapter is based on unpublished original work.

Regarding the structure, the chapter starts with a selection of presentation and visualisation methods applied in this case (reifying the discussion in section 2.3.4), complemented by another section dedicated to model verification, where the tools and approach applied in this context are outlined. Within the subsequent section on experimentation and simulation scenario generation, results of the simulation are exemplified, followed by a section on interpretation of these results, which is the basis for model validation.

The order of the sections of this chapter basically reflects the procedure (i.e. the sequence of steps, together with the prerequisites) applied during and after the implementation of the simulation model.

7.2 Presentation of simulation results

The presentation of data produced by simulation runs is crucial to get the most out of the modelling endeavour, as such simulation outcomes are the basis for the subsequent — and in many cases necessary — analysis and interpretation of simulation results. There are many different ways to condense and present data. Firstly, the kind of data (qualitative — text logs, quantitative — numbers) is a decisive factor for selecting the presentation method; secondly the intended analysis method constrains the representation, and last but not

least preferences of the persons reading and analysing the data have a strong influence.

Basically, thoughts have to be given whether

1. to present just raw data generated by simulations, and/or
2. to pre-process, condense and/or visualise the data already during simulation runs.

Usually both ways should be followed to some extent, as the 1st is needed to do analyses a posteriori (also of kinds not originally planned beforehand), while the 2nd is primarily useful in the state of model verification to observe unexpected or erroneous model behaviour, but also for validation and result documentation. For example, snapshots from animation sequences can be used to illustrate and highlight certain simulation result in model-based scenarios.

In the following a selection of the methods used to present the results of the use case model are explained. Before, however, some related terms regarding output data from simulation software must be defined, as used in this chapter and other parts of this PhD thesis:

- *Simulation outcomes* are denoting (raw) data generated by simulation runs.
- *Simulation results* is information gathered from simulation runs, mainly (automatically or manually) produced by analysing, condensing or otherwise processing simulation outcomes.
- *Model-based scenarios* are narratives (sometimes called 'vignettes'), bringing together simulation results of various kinds and formats, providing an integrated view on a specific context, and are manually written in a language easily graspable by the envisaged target group.

7.2.1 Text logs

Due to the qualitative evidence the model is based upon, the most important outcome is the text log, i.e. statements generated by rules that fired in connection with meta-information (name of rule, tick/time stamp of firing, traceability information etc.). This log is a kind of story generated by the model, documenting the events that happened in this particular model execution in a (chrono-) logical order. These simulation logs are the basis for further analysis, and hence written in raw fashion both to result files and to consoles on-the-fly. Figure 7.1 shows an example of such a log console.

7.2.2 Graphical visualisation

A drawback of textual logs is that the log entries are presented from all sources in one view, e.g. all statements produced by rules in one tick are shown in one

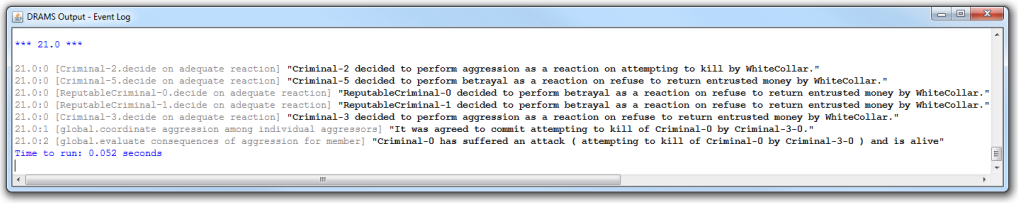

Figure 7.1: DRAMS output window for event log

block. Even if filters are applied to display only entries with certain parameters — e.g. produced by one particular agent — it is still difficult to capture a really crucial bit of information: interactions between agents, in particular when encompassing several simulation ticks. In order to give an easy visual access to this kind of information, the logs are translated into a graphical representation, resembling 'comic-like' sketches of the agents with 'text bubbles' showing thoughts, actions and interaction, together with optional visualisation of different kinds of relations between the agents. These relations are drawn as grey arrows and can carry different information:

- The individual image a criminal agent has of a fellow criminal agent is visualised by the thickness of the connecting edge. In this case, a full-meshed network is drawn, where just the width of the connecting lines carries the relevant information.

- The emergence of 'authority' is shown by edges, e.g. the raise of image is visualised by an arrow from the agent which raises the image, pointing towards the target agent.

A screenshot of an early state of the simulation (the initial normative event with possible individual reactions) is shown in Figure 7.2 (more examples are given in Figure 7.13 and 7.14). All criminal agents (represented by actor symbols) are arranged in a circle; on the right-hand side are icons for the police agent (blue hexagon) and the general public (cloud-shaped). The filling colour of the criminals reflects their kind (grey for Black Collar, white for White Collar), the outline colour gives a hint about the current mental frame of the agent (dark grey for rational frame, red for emotional frame). The text bubbles with related arrows are coloured in bright hues, like in this example a light yellow. For each new external normative event and also in the case of (individual) panic of an agent, a new colour is chosen in order to allow following such trails.

Circular bubbles show normative events, sharp-edged rectangles represent actions, and with round-edged rectangles reasoning or other intra-agent matters are made visible. These text bubbles are connected via arrows to the agent from which the information is emitted, while from action bubbles also

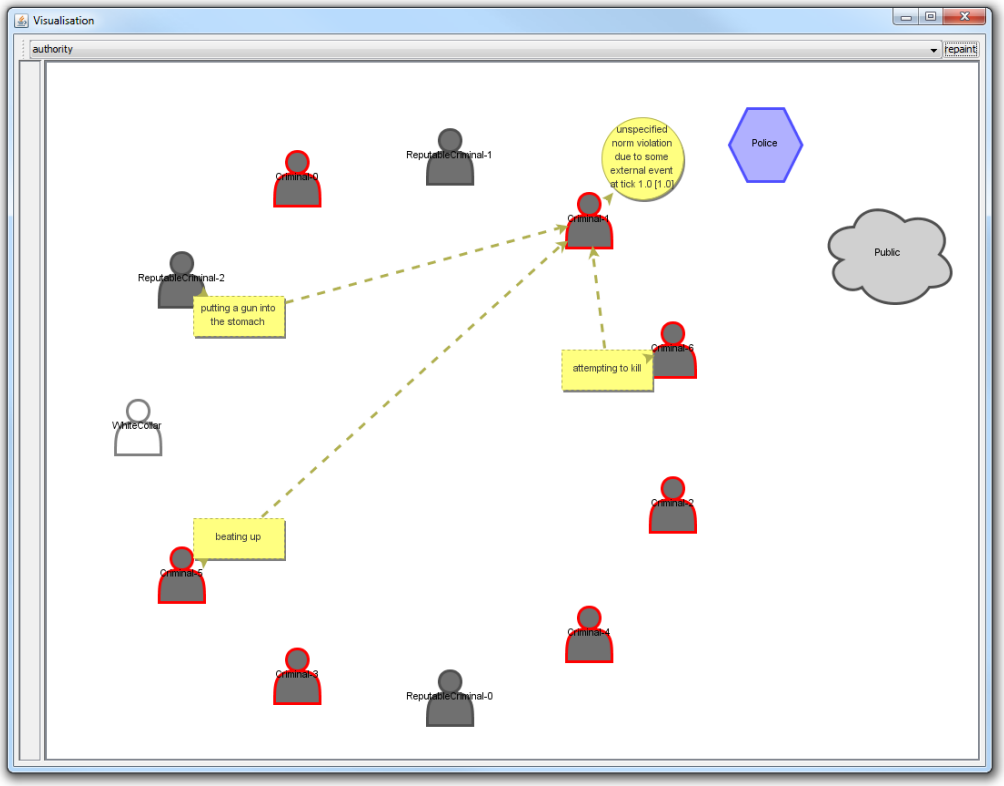

Figure 7.2: Model visualisation window showing the first reaction to an external event

the designated target is delineated with an arrow. Last but not least, the line style of bubble outlines and arrows gives an indication of the state of the action: if dashed, then just a plan for an actions is made; if solid, then the plan is carried out.

7.2.3 Value tables and time lines

The progression of image values is a central gauge of the state of simulation. Therefore, both the individual values (i.e. the image each agent has of all of its fellow agents) as well as average image of each agent are written to value tables. For this purpose, comma separated value (CSV) tables are applied. As an example, Figure 7.3 contains such a table with image values of agent 'ReputableCriminal-1' over the course of a simulation run. According to the model design, the image is not represented by number but by ordinal-scaled literals: very low, low, modest, high and very high.

In order to have a measure for average image values, the five individual ordinally-scaled literals have to be converted to matching numbers (from -2 to +2).

	A	B	C	D	E	F	G	H	I	J	K	L
1	tick	WhiteCollar	Criminal-0	Criminal-1	Criminal-2	Criminal-3	Criminal-4	Criminal-5	Criminal-6	ReputableCr	ReputableCriminal-2	
2	0	veryhigh	modest	modest	modest	modest	modest	modest	modest	high	high	
3	1	veryhigh	modest	modest	modest	modest	modest	modest	modest	high	high	
4	2	veryhigh	modest	modest	modest	modest	modest	modest	modest	high	high	
5	3	veryhigh	modest	modest	modest	modest	modest	modest	modest	high	high	
6	4	veryhigh	modest	modest	modest	modest	modest	modest	modest	high	high	
7	5	veryhigh	modest	modest	modest	modest	modest	modest	modest	high	high	
8	6	veryhigh	modest	modest	modest	modest	modest	modest	modest	high	high	
9	7	veryhigh	modest	modest	modest	modest	modest	modest	modest	high	high	
10	8	veryhigh	modest	modest	modest	modest	modest	modest	modest	high	high	
11	9	veryhigh	modest	modest	modest	modest	modest	modest	modest	high	high	
12	10	veryhigh	modest	modest	modest	modest	modest	modest	modest	high	high	
13	11	veryhigh	modest	modest	modest	modest	modest	modest	modest	high	high	
14	12	veryhigh	modest	modest	modest			modest	modest	high	high	
15	13	veryhigh	modest	modest	modest			modest	modest	high	high	
16	14	veryhigh	modest	modest	modest			modest	modest	high	high	
17	15	veryhigh	modest	modest	modest			modest	modest	high	high	
18	16	high (arbitraryAggression)	modest	modest	modest			modest	modest	high	high	
19	17	high	modest	modest	modest			modest	modest	high	high	
20	18	high	modest	modest	modest			modest	modest	high	high	
21	19		modest	modest	modest			modest	modest	high	high	
22	20		modest	modest	modest			modest	modest	high	high	
23	21		modest	modest	modest			modest	modest	high	high	
24	22		modest	modest	modest			modest	modest	high	high	
25	23		low (arbitraryAggression)	modest	modest			modest	modest	high	high	
26	24		low	modest	modest			modest	modest	high	high	
27	25		low	modest	modest			modest	modest	high	high	
28	26		low	modest	modest			modest	modest	high	high	
29	27		low	modest	modest			modest	modest	high	high	
30	28		low	modest	modest			modest	modest	high	high	
31	29		low	modest	modest			modest	modest	high	high	
32	30		low	modest	modest			modest	modest	high	high	
33	31		low	modest	modest			modest	modest	high	high	
34	32		verylow (arbitraryAggression)	modest	modest			modest	modest	high	high	
35												

Figure 7.3: Spreadsheet table for individual image values of ReputableCriminal-1 for all fellow criminals over 32 ticks

In addition, the average image values are printed to a time line diagram as shown in Figure 7.4. Further time line diagrams exist for the numerical variable 'number of agents' as well as for the ordinally scaled (similar to image) reputation of agents. The condensed information displayed in these diagrams are a very useful means to identify pattern in simulation runs. For example, a rapidly decreasing number of active criminals either indicates an extremely violent breakdown or extraordinarily successful police actions. A look at the image diagram typically allows for a further distinction: massively falling average image values most often also signal excessive violence.

The time line diagrams are drawn on-the-fly during the simulation runs and, thus, allow to monitor and potentially classify the model execution.

7.2.4 Sequence diagrams

The text logs (together with the associated meta data and the value tables for the different variables) contain all the information generated by the model. However, it can become difficult to keep an overview on the actual course of events, their preconditions and effects just from looking at the text and diagrams. Hence, sequence diagrams are produced that contain exactly this information, allowing access to the overall simulation run as well as to all the particular strands within the run. These strands (the same that are distinguished by different colours in the animation) are defined by a normative event

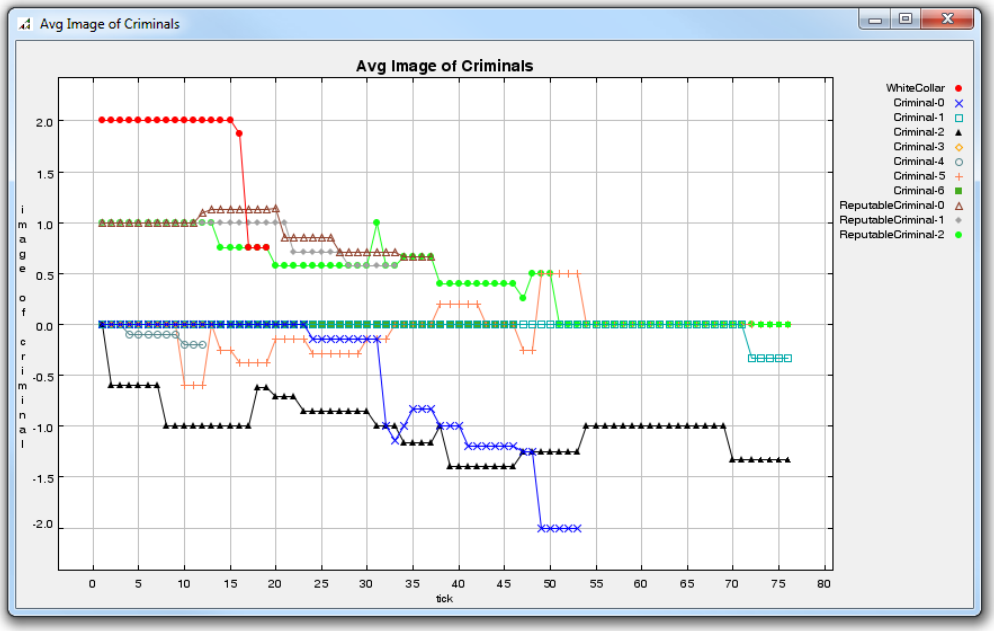

Figure 7.4: Time line diagram showing the progression of average image values of the members of the criminal network

as a starting point, from which reactions and further events might occur. Such sequence diagrams tell 'independent stories', which even can happen in parallel with other stories. These diagrams easily allow to identify the cause for an observed action in a simulation run. They can easily be translated by a human into short text fragments, which can be compiled later into a complete model-based scenario, covering the entire run. Figure 7.5 gives an example of such a strand where WhiteCollar is intimidated by Criminal-0. With some elements of interpretation, this sequence diagram can be transformed into the following story:

> Criminal-0 becomes aware of the arresting of Criminal-4 and gets into a panic that the next police action could be against WhiteCollar, making it impossible to get back the money he gave him for money laundering. He goes to WhiteCollar and asks for money, at this occasion not only for his own share but for a much higher amount. WhiteCollar is unable to pay such a high amount, and completely refuses the request. This brings Criminal-0 into a rage (as such behaviour is a violation of the norm of trust) and he vents his frustration by starting an affair with the girlfriend of WhiteCollar. This, in turn, infuriates WhiteCollar so much that he decides to kill Criminal-0, but fails (for he is not an experienced gunman). Criminal-0 plans to retaliate in the same way. He

chases up WhiteCollar, and when he faces him with his drawn gun, WhiteCollar starts begging for his life and regrets his misbehaviour, so Criminal-0 puts away his weapon. ReputableCriminal-1, who is witness of this situation, appreciates the obedience to the norm of trust by WhiteCollar. The conflict has calmed down for the moment.

As a side remark, the sequence diagram shows two notable peculiarities of the model:

1. There are two acts of obeying by WC at tick 36 visible. This is due to two norm violations in the past of WC which he relates to the received sanction (in the normative process).

2. The image of ReputableCriminal-1 towards WhiteCollar shows a counterintuitive increase from 'very high' to 'very high'. This means that the image actually does not change (because it is already at the topmost level), but this event is important anyway, as it can trigger actions even when its value is already 'very high'.

Summing up, each simulation run produces the following artefacts:

- Text log files in plain text and csv format, as well as an xml log file containing additional traceability meta-data (files `events.txt/csv/xml`).

- Tables in csv format with individual image values for all criminal agents (files `imageTable_[NameOfCriminal].csv`).

- Table in csv format with average image values (file `imageTableAvg.csv`).

- Time line diagrams (snapshots from the on-the-fly diagrams) for number of criminals (`graphNumberOfCriminals[tick].png`), average image values (`graphAvgImageOfCriminals[tick].png`) and reputation values (`graphReputationOfCriminals[tick].png`).

- A full sequence diagram for the entire simulation run (file `sequence_full.uxf`[91]). This diagram is built up successively during the run and can be accessed and displayed at any time (file `sequence_part.uxf`).

- Sequence diagrams for the different strands of actions (files `sequence_strand-[strandNumber]_[startingTickOfStrand].uxf`).

- Snapshots of the graphical visualisation for each simulation tick (files `vis-tick_[tick].png`).

[91]The uxf format is an xml-based format that can be displayed with the UMLet software, `http://www.umlet.com`

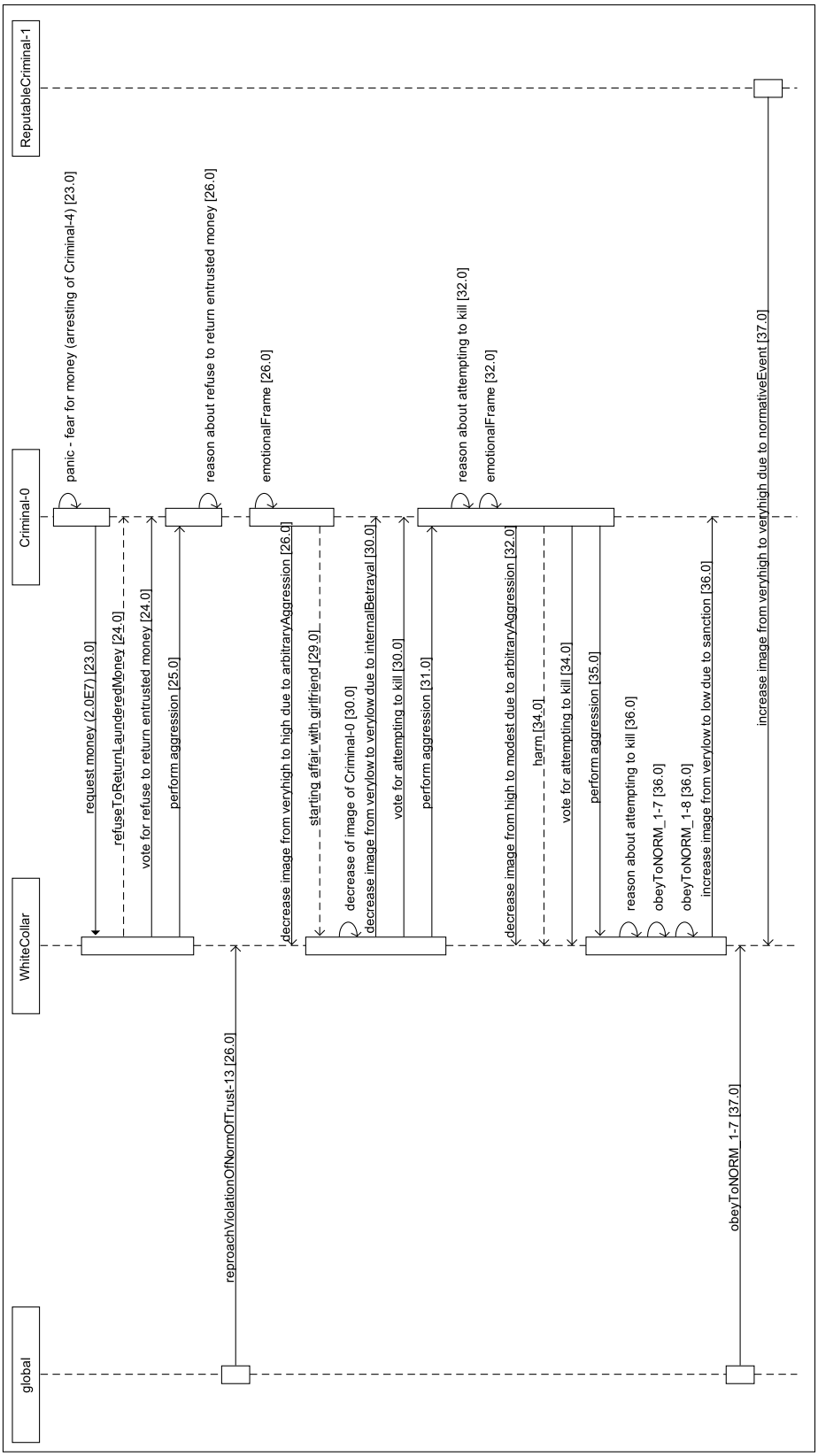

Figure 7.5: Model-generated sequence diagram with a particular strand of action, starting with a panic reaction of Criminal-0

7.3 Simulation model verification

The first time the presentation of simulation outcomes becomes relevant is the model verification phase. This phase starts with code 'debugging' already early during implementation, and concludes with the endeavour to verify that as many as possible relevant aspects contained in the CCD (ideally all of them) as well as additional requirements are implemented in a formally correct way[92]. For this use case, the process outlined in section 4.4.2 is applied. In each expansion stage of the implementation (on the level of single rules as well as sets of rules tackling a certain aspect of the model) the code is checked with test facts to verify the intended behaviour. Occasionally a rule or model segment shows unexpected behaviour, caused either by incompletely or inconsistently defined conditions, or by unexpected side effects, not rarely due to the particular data driven way of rule evaluation by DRAMS. In the former case, adaptations to the CCD and repeated code transformation are often a clean way to solve the issues.

The presentation means introduced in the previous section are mainly intended for productive runs (i.e. to generate data to be analysed and interpreted in simulation results), but also play an important role to support the verification. In addition to that, specialised instruments for code inspection can be crucial for model verification. DRAMS provides several user interface components in this respect. These will be introduced in the following subsections.

7.3.1 Model Explorer

The process of exploring a model subsumes different tasks with the aim to gain insights into structure and behaviour of the model — two usually interrelated aspects. While primary interest is to understand the behaviour of the model, it becomes necessary to also comprehend the structure of the process leading to this behaviour, and to present these insights in different levels of abstraction dedicated to the targeted users. While for example the model programmer wants to know details about each rule that fired together with any data processed by the rule, a domain expert (or, more generally, a stakeholder) for instance might only want to see an overview of different fact types and how these are related to the agents representing real-life actors. Hence, the Model Explorer takes these aspects and perspectives into account.

The detail of the screenshot in Figure 7.6 is captured from the DRAMS Model Explorer, a plugin to visualise the rule processing and traceability features of DRAMS by exploiting internal data structures of the rule engine. In the picture, a small part of the DRAMS evaluation tree (just one rule with three pre-conditions) is visible at the top, where a pop-up window is displayed for a link from one of the pre-condition facts to the rule. This pop-up info box

[92] In context of verification, correctness is only checked against the specification, in particular against the CCD. Further checks 'against reality' are subject of validation later on.

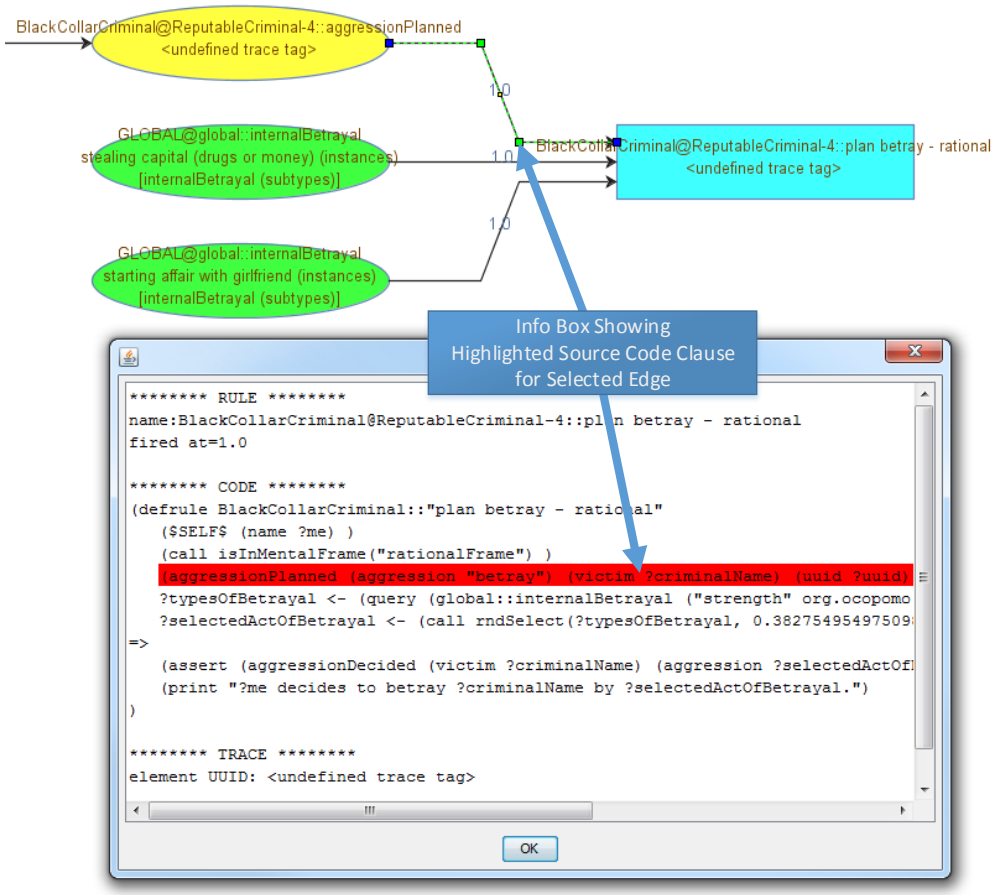

Figure 7.6: DRAMS Model Explorer showing rule evaluation tree with debug information for selected condition

contains the rule code, while highlighting the clause to which the link belongs. Among others, the source code of the rule is displayed, with the particular retrieve clause highlighted. Info boxes for facts contain information about the concrete content of the facts as seen by any succeeding rules, while boxes for fact templates reveal the facts available for the template. In the context of model verification, the localisation of causes for unexpected (and potentially erroneous) model behaviour is simplified since not only data (i.e. fact content) but also the model structure and source code can be inspected at the same time in an integrated view.

From a modelling expert perspective, graphically tracing the rules in the evaluation tree is not only useful for debugging, but also for helping implementers newly confronted with the model to understand existing code.

The primary purpose of the DRAMS Model Explorer is, however, the visualisation of traceability information relevant for domain experts. Figure 7.7

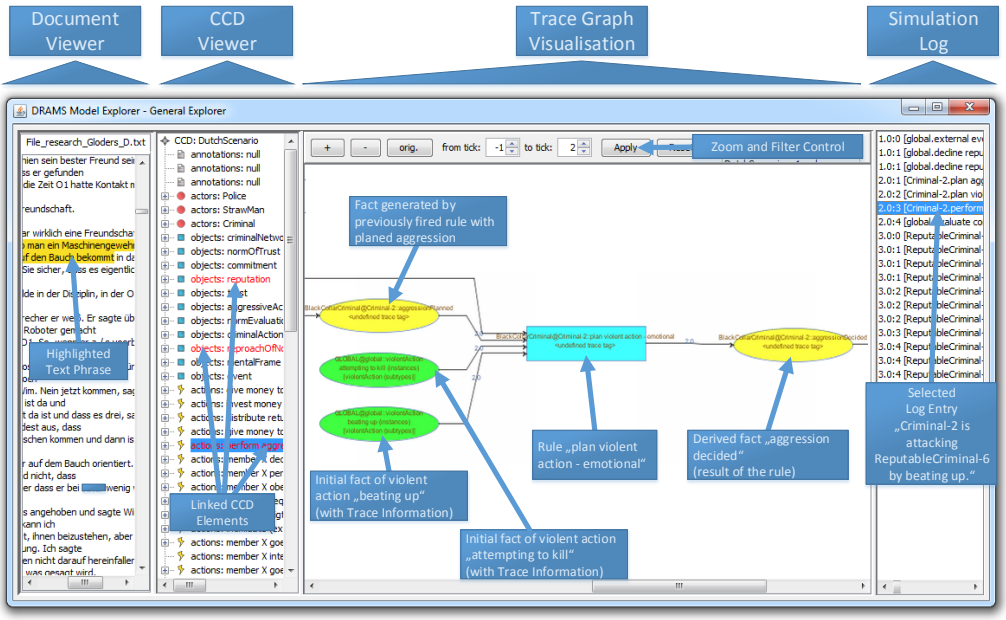

Figure 7.7: DRAMS Model Explorer user interface with traceability visualisation support

contains a screenshot of the full Model Explorer user interface. Four views can be distinguished in the window:

- The right-hand side shows the simulation log, i.e. the outputs generated by the rules.
- In the centre, the evaluation tree is calculated and visualised for a particular log entry. If the user selects one (or a set) of the entries, the associated tree is displayed. A filter allows to restrict the number of ticks included in the evaluation tree.

  The green ovals on the left are initial facts; the yellow ovals represent facts asserted during the simulation run. The blue box between the facts is the rule that uses the (left) facts as pre-condition and asserts the (right) fact as post-condition. The two initial facts have trace tags attached.
- Left to the evaluation tree, the CCD is displayed. All elements that have traces in the currently displayed evaluation tree are marked.
- On the very left, the evidence documents can be found. Here the text annotations linked with the marked CCD elements are highlighted.

Technical details on the Model Explorer and trace generation by DRAMS are given in Chapter 10.

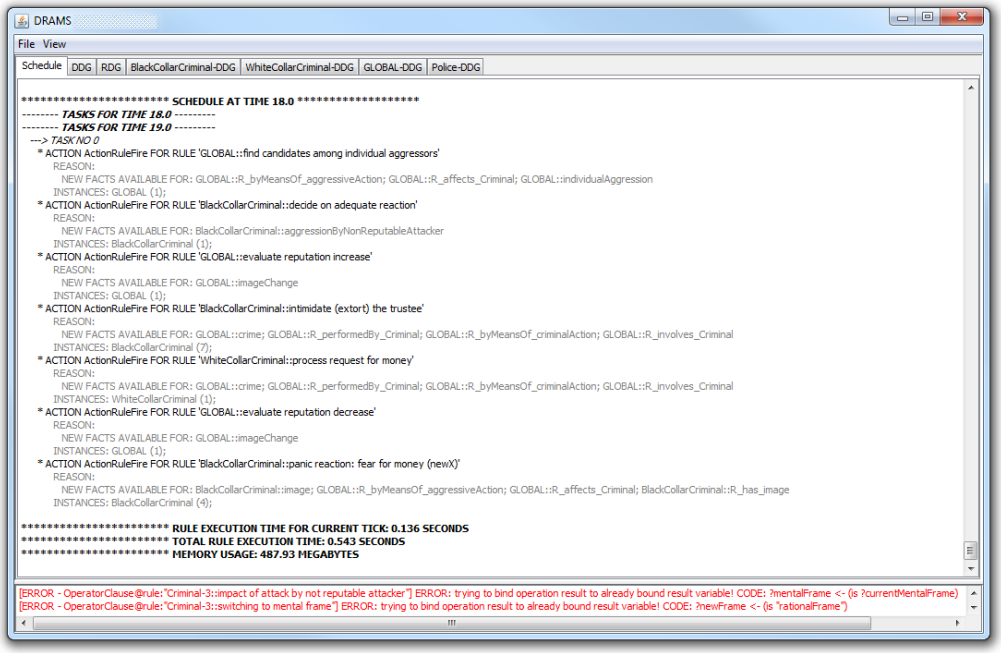

Figure 7.8: DRAMS rule schedule

7.3.2 Rule Schedule

On a technical level, access to internal data structures of the simulation system can be helpful for debugging and verification. In case of the DRAMS UI, a number of views are available in this context.

Firstly, a rule schedule can be investigated e.g. if a rule fires unexpectedly, or a rule expected to fire does not. The rule schedule is derived from the data driven pre-evaluation of rules for which all necessary pre-conditions are fulfilled purely from the data perspective, i. e. new facts are available in the fact bases that are explored by the LHS's of the rules. From this information it cannot be concluded that the rules actually fire (i.e. the RHS's are executed), since the content of the LHS facts are evaluated in a different step. Thus, this list contains all rules for which the actual evaluation will be triggered by DRAMS. An example of the rule schedule view can be seen in Figure 7.8.

7.3.3 Console

If a rule is scheduled for evaluation but does not fire although all conditions seem to be valid, a look into the fact base can help to narrow down the problem. A fact base dump (with full content or filtered for an agent instance, if required) is contained in the DRAMS console shown in Figure 7.9. This DRAMS UI component can also be used to execute (draft) rules 'on-the-fly', based on the

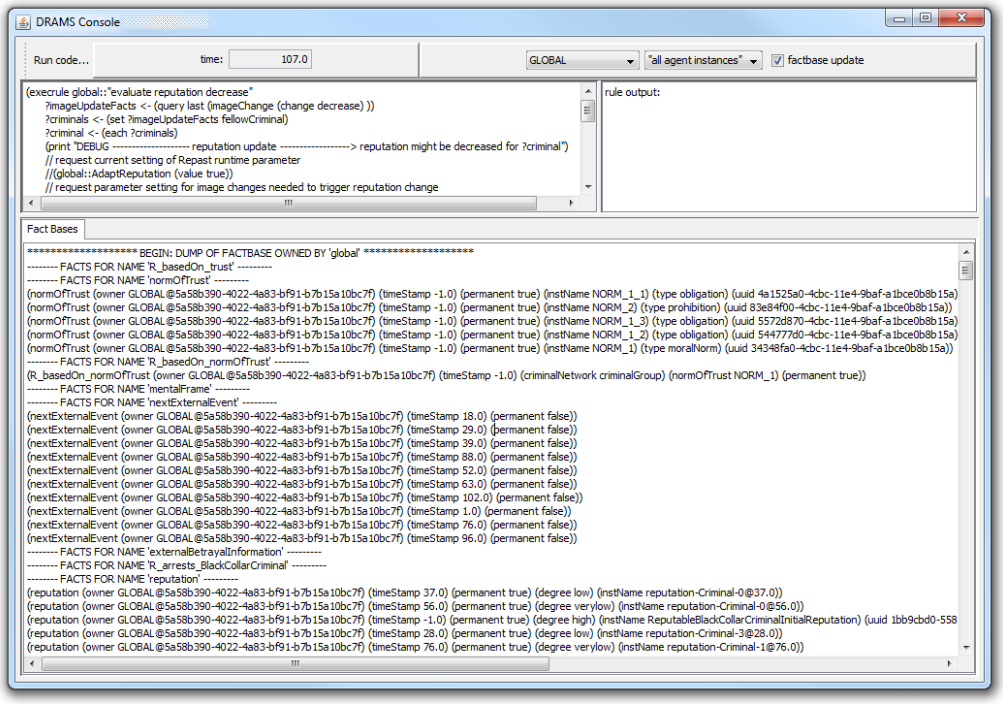

Figure 7.9: DRAMS on-the-fly rule execution console with fact base explorer

current tick and state of the fact bases (as initially set or generated by other parts of the model already implemented). This is helpful for both debugging and drafting/editing new rules.

7.3.4 Dependency Graphs

In order to check the correctness of dependencies between rules, the Rule Dependency Graph (RDG) diagram can give a first overview (Figure 7.10). For a particular state of the rule engine (in this case the initial state) a directed graph is generated containing all rules that might fire given the current fact base configuration. This can change during simulation, as new facts are generated by firing rules.

More detailed views are available in the Data Dependency Graph (DDG) diagrams, where all rules with the related fact templates are displayed. The actual fact base state (defined by the asserted facts) has no influence on the number and structure of elements shown, but it is reflected in the colour of the oval fact template symbols:

- For green fact templates actual fact instances are initially set (i.e. facts asserted via DRAMS code).

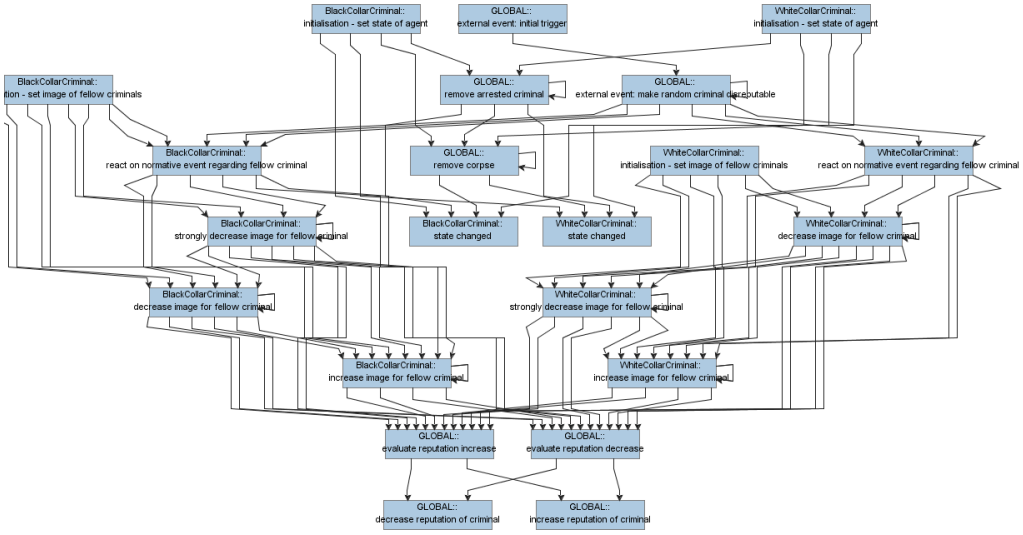

Figure 7.10: DRAMS Rule Dependency Graph (RDG) at the start of model execution

- Red fact templates indicate that no initial facts are available, but are asserted by rules during the simulation run.

- There is also the case that a fact template used for retrieving or asserting facts is unknown at initialisation time due to dynamic fact base selection (i.e. the fact base to use is determined during rule processing). Here the symbol is drawn in yellow colour.

Views are available for the full DDG and individual (smaller) DDG fragments for each agent. While the full DDG can help to check inter-agent relations, the latter allow for a quick view inside intra-agent processes. Figure 7.11 gives an example with a part of a BlackCollarCriminal DDG, filtering the view on just one rule and the adjacent facts for pre- and post-conditions.

The dependency graph diagrams are — together with the rule schedule — part of the main DRAMS UI window. As visible in both Figure 7.11 and Figure 7.8, the additional error log view ath the bottom can become handy at times.

7.4 Experimentation and simulation scenario generation

With a verified simulation model and the means for generating simulation outcomes in various formats, the experimentation phase can start. Although the model has a strong stochastic element, the intention is not to consider

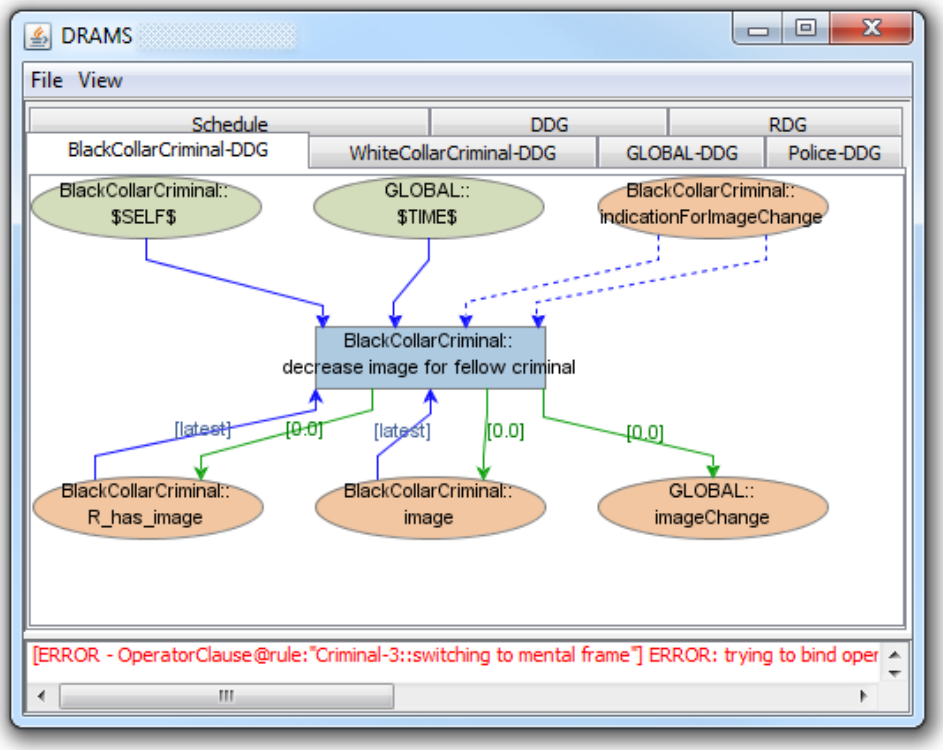

Figure 7.11: DRAMS Data Dependency Graph (DDG) of Black Collar Criminal, filtering on rule 'decrease image for fellow criminal'

quantitative analysis (at least) at the beginning, but rather let the model generate stories, i.e. mainly qualitative information. The rationale behind this approach can be summarised as follows:

- Despite the 'noise' produced by the multitude of stochastic processes on input variables, the expectation is to be able to recognise limited numbers of behavioural patterns in the outcomes.
- The qualitative (i.e. textual) outcomes would allow to easily write model generated scenarios in a language familiar to the stakeholders, by drawing on the evidence base by using traceability features of the toolbox. These scenarios would point out different paths of action with associated root causes and influence factors, and are a kind of "virtual experience"(Lotzmann and Neumann, 2016) to the stakeholders.
- Scenarios written for the identified pattern would be adequate means to firstly check for plausibility, and secondly to approach the stakeholders and discuss the findings. These discussions could help to transform (more

and more) stochastic input variables into deterministic decisions, 'hardening' the model against influence factors or (newly introduced) events or parameters, with reference to the next point:

- Prospectively, to use the model — validated by the procedure latent in the previous points — for further research questions, which might even involve quantitative analysis.

The experiment design applied in this use case caters for the aforementioned intentions. Only a very small set of parameters are varied, initially only the proportion of reputable and non-reputable Black Collar criminals, while the total number of Black Collar criminals remains constant (10). When the found patterns are analysed and understood, further parameters can be varied to study the influence to the known pattern. Additional parameters include (see parameter setting user interface in Figure 7.12):

- Repeated external normative events, i.e. making members of the criminal network disreputable for unknown reasons.
- Taking the influence of sanction to image into consideration: If a victim of an aggression recognises this as a justified sanction, the image of the aggressor is changed to a higher value.
- The adaptation of reputation to image change: If the image of an agent changes for a certain number of times exclusively in one direction, the reputation value is adapted proportionally[93].

As mentioned earlier, by having stochastic elements in many decisions, a wide behaviour space of simulation outcomes is generated. However, as expected based on the model design, this behaviour space contains a limited set of behavioural patterns. In the first place, these patterns can be identified by the 'final' state. Final state is defined here as a state that remains stable over a certain number of ticks without any state change in the model caused by individual agent behaviour. If a further analysis reveals that different paths of actions are causing similar final states, the differences have to be captured by additional patterns. It is important to clearly identify the 'turning points' or 'critical junctures' (Lotzmann and Neumann, 2016), i.e. events that are pivotal for (emerging) shapes of behaviour. To identify (or confirm) such turning points as 'milestones' in the pattern, it is necessary to observe and analyse a large number (in the magnitude of tenth to hundreds) of simulation runs.

In the end, one typical simulation run for each pattern is picked out as basis for the model-based scenario. Writing a scenario can closely follow the generated simulation logs and sequence diagrams (as shown above in the text belonging to Figure 7.5) and should make use of meta-information available

[93]This is an experimental feature for reputation change as mentioned in the outlook of section 6.10

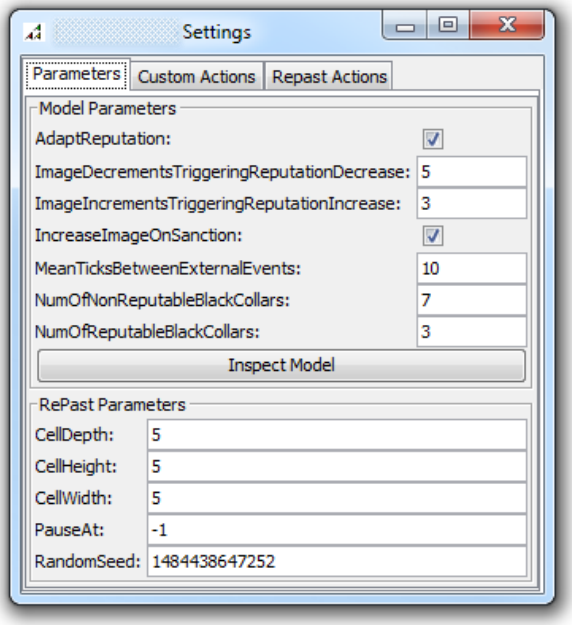

Figure 7.12: Repast user interface window for model parameter settings

and visualisations where suitable for illustration.

A structured template can also become handy for writing a model generated scenario from outcomes of a simulation run. Collecting the 'interesting' information from the different outcome presentations, condensing them into small text fragments for each tick (or a small number of ticks covering interrelated content) and adding relevant traceability information (phrases and annotations, identified e.g. with help of the DRAMS Model Explorer) like done in Table 7.1 is a recommendable first step for documenting experimentation results.

The table contains a condensed summary from a simulation run, where the criminal network experiences a rapid and violent break-down. It follows a typical pattern of simulation outcomes that resembles the fate of the real criminal network behind the evidence documents, in this case in a particular violent manifestation. The network in this simulation is destroyed in just 38 ticks, it involves many parallel strands of conflicts, which quite well reflects the "corrupt chaos" as reported in interrogations from the real case. The critical junctures in this run are:

- An initial murder at tick 6 as result of a bilateral conflict. This sets off a panic among the criminals.
- Betraying the network by informing the public (tick 17) and the police (tick 21) causes investigations by the police.

- Arresting of Criminal-4 at tick 22. This starts an intimidation of WhiteCollar, and constitutes the phase of terminal decay of the gang.
- Arresting of WhiteCollar (tick 38) drains the financial basis for the remaining network members.

Already at this stage of description, a first interpretative element is applied on the condensed summary in Table 7.1, which is human written, but still near the output of the model. For example, the statement that Criminal3 decides in *rage*, is derived from the fact that ordinary (non-reputable) criminal agents are initialised to act in emotional mental frame. This way of interpretation becomes more important in the final stage of generating qualitative simulation results: writing of the model-based scenarios. This transformation is demonstrated in the following section, where also the actual results from simulation experiments are discussed, including the second step of interpretation in a style resembling a short story.

Table 7.1: Scenario derived from a simulation run

Tick	**Summary of log entries**	**Annotations and traceability information**
2	ReputableCriminal-0 becomes susceptible due to unspecified norm violation. This is observed by Criminal-3 who decides in rage[94] to murder ReputableCriminal-0.	*Now you'll die.*
3	ReputableCriminal-0 reasons about aggression: Since Criminal-3 has a rather bad reputation he switches into panic mode.	
4	ReputableCriminal-0 decides to react with aggression.	
5	Attempt to kill as counter-aggression.	*An investigation against the Hells Angels revealed that presumably V01 asked the Hells Angels to make an operation against O1 in return for a huge amount of money.*
6	Murder successful, Criminal-3 dead.	

Continued on next page

[94]Criminal-3 is per definition initially in emotional mental frame.

Table 7.1 – continued from previous page

Tick	Summary of log entries	Annotations and traceability information
7	The murder is observed by Criminal-1, 2, 5 and 6 as well as ReputableCriminal-1. They switch in panic mode: fear for life. Criminal-6 tries to kill Criminal-4 as reaction, and Criminal-2 tries to kill Criminal-5. Criminal-2 has furthermore become suspicious to have violated the norm of trust (possibly in another matter), so Criminal-5, 6 and ReputableCriminal-1 decide together to kill Criminal-2.	
8	Criminal-5 is dead, the other murder attempts were unsuccessful.	
9	The attacked agents try to find a reason. Criminal-4 cannot see the aggression as a sanction as the attacker Criminal-5 also has a bad reputation. Criminal-2 regards the aggression by ReputableCriminal-1 as a potential sanction but cannot see its own norm violation. All agents except Criminal-2 and ReputableCriminal-0 get into a new panic due to murder of Criminal-5, which leads to (independent) murder attempts against Criminal-0 and ReputableCriminal-2.	*There is a corrupt chaos behind it.*
10	ReputableCriminal-2 is dead. Criminal-0 survived.	

Continued on next page

Table 7.1 – continued from previous page

Tick	Summary of log entries	Annotations and traceability information
11	New panic (fear for life), triggering new aggressions: beating up, outburst of rage and again attempting to kill.	Annotation 1: *For the first time I saw that he had lather in his mouth and kicked a bicycle against a tree.* Annotation 2: *His head was completely deformed, his eyes blue and swollen.*
12	ReputableCriminal-0 and Criminal-6 are dead.	
13	Panic, no sanction or own norm violation found by surviving victims of aggression. New aggressions against Criminal-1, 2 and ReputableCriminal-1, furthermore WhiteCollar tries to kill Criminal-0.	*He was at a point where he was totally despaired. He had a plan to approach O1 with a weapon. However, in the last moment he didn't dare.*
14	Criminal-1 dead.	
15	Criminal-0 survives attack by WhiteCollar, and regards this as a justified sanction. Criminal-2 and ReputableCriminal-1 are going to betray the criminal network.	
16-17	Both are going to public.	Annotation 1: *The affair with M. had been in the newspapers.* Annotation 2: *O1 wanted the money from her and threatened M. to kill him and his children. M. told the newspapers [about my role in the criminal network] because he thought that I wanted to kill him to get the money.*
18	Criminal-2 obeys the norm of trust (aggression by ReputableCriminal-1 is regarded as sanction), but also new aggressions among the remaining members.	

Continued on next page

Table 7.1 – continued from previous page

Tick	Summary of log entries	Annotations and traceability information
19	New aggressions and violence, hence Criminal-4 obeys the sanction.	
20	New aggressions and act of betrayal.	
21	ReputableCriminal-1 informs the police, but also tries to kill Criminal-2, and further violence.	*On Aug. 8, [...] the criminal intelligence of [the city of A.] received an anonymous letter.*
22	Criminal-4 arrested.	
23	Criminal-0 and 2 get into a fear for money due to the police action. They try to get their money from WhiteCollar.	
24	White Collar pays the money to Criminal-2, the amount requested by Criminal-0 was too high, so the payment was refused. The conflict with ReputableCriminal-1 leads Criminal-2 to betray the network.	
26-29	Criminal-0 starts to intimidate WhiteCollar by starting an affair with the girlfriend of WhiteCollar.	
30-34	Violent conflict between WhiteCollar and Criminal-0, that WhiteCollar resolves by obedience in later tick (see Figure 7.5).	
35	ReputableCriminal-1 sets off to kill Criminal-0.	
36	Criminal-0 dead. New panic.	
37	New intimidation of WhiteCollar.	
38	WhiteCollar arrested by police.	
39	Final state: Two remaining (active) members of the criminal network (Criminal-2 and ReputableCriminal1), two arrested members (Criminal-4 and WhiteCollar).	

7.5 Interpretation of simulation results and validation

As stated in (Lotzmann and Neumann, 2016), the model-based scenario closes "the cycle of qualitative simulation, beginning with a qualitative analysis of the data as basis for the development of a simulation model and ending with analyzing simulation results by means of an interpretative methodology in the development of a narrative of the simulation results." Writing such a scenario can be an involved endeavour, in particular for long runs (over many ticks) with many parallel strands of action. While staying close to the raw text logs, even when condensed and otherwise preprocessed like in Table 7.1, it still takes significant effort to extract and intersperse matching traces to the evidence base, to take values of variables (like image and reputation) into account, and to find natural-language replacements for 'unemotional' and artificial-sounding compositions of log entries printed by the rules. The key here is to interpret the different outcomes from an overarching viewpoint, to underpin events and actions by meaning taken from the evidence base. The overall aim of this process is to find a language similar to the documents provided by the stakeholders, and to match the domain language of the stakeholders in general.

The following paragraphs present an attempt to formulate such a scenario from the content of Table 7.1, a 'criminal short story' about the demise of a criminal network due to internal conflicts. The phrases in italics are quoted verbatim from the evidence files.

> The drama started with a gossip. For reasons unknown to the observer, Reputable Criminal (RC) 0 became susceptible. A delivery of drugs was incomplete, and there were talks that RC 0 snatched some for own consumption. This would have been a massive violation of the norm of trust, but it may not be so and just some bad talk behind the back. However, at least Criminal (C) 3 believed in the bad talk and felt himself offended by RC 0 so much that he got into a rage. He rushed to RC 0, approached him with his drawn weapon and confronted him hysterically with his accusations, repeatedly yelling "*Now you'll die*". In the emerging turmoil the gun went off, but the bullet missed RC 0. RC 0 ran for his life. After he recovered a bit from this shock, he started reasoning over what just happened — to be attacked by this crude scumbag C 3. The more RC 0 thought about it, the more he got into a panic.
>
> RC 0 hatched a plan to retaliate and eventually hired an *outlaw motor cycling gang to make an operation against C 3 in return for a huge amount of money*, who did a successful drive-by shooting,

leaving C 3 dead on the sidewalk. As C 1, C 2, C 5, C 6 and RC 1 got notice of that assassination they panicked. Not knowing who was responsible for this attack they got in fear for their own lives, and they attempted to counter-react by trying to eliminate the perpetrator. Each of them made wild guesses whom to blame as commanding the attack. While C 1 was like paralysed and not able to do anything, C 2 tried to kill C 5 as reaction, and C 6 tried the same with C 4. In addition, C5, C6 and RC 1 ganged up against C 2. However, only the attack against C 5 ended with the intended result, the other attempts had been planned in a hurry and failed. Both C 2 and C4 survived the attacks. As neither of them commanded the execution they were sorely shocked as they found no explanation for the attack.

Due to the death of C 5, the remaining members of the gang (except RC 2 and C 2) were affected by a new wave of panic, as they felt that *there was a corrupt chaos behind it.* It did not take long before reactions took place: RC 2 was murdered, C 0 barely survived an attack.

A new series of violence struck the remaining gang members (Figure 7.13). In an outburst of rage against C6, *C 1 had lather at his mouth and kicked a bicycle against a tree.* Both of them fell victim of a gun attack by other gang members, first C 6, later also C 1. C 2 however, with brute force beat the hell out of RC 1 until he was fit for the hospital. *His head was completely deformed, his eyes blue and swollen.* Another deadly attack against RC 0 followed. Also the White Collar (WC) who (as the reputable money launderer) kept his nose out of the trouble so far *was at a point where he was totally despaired. He had a plan to approach C 0 with a weapon. However, in the last moment he didn't dare.* But this planned attack became known to C 0 and had a result: He regarded this as a justified sanction, a word of command by WC. Violence started to cool down a bit, but the panic was not over: C 2 and RC 1 tried to get even with the former fellows and now foes by betraying each other and the remaining gang members: suddenly a report about a fictitious *affair about RC 1 had been in the newspapers* triggered by C 2, while RC 1 *gave an interview with the newspapers in which he explained the role of C 2 in the criminal network.*

From time to time violence flared up again, until RC 1 searched his way out of trouble by informing the police: *On Aug. 8, the criminal intelligence received an anonymous letter.* The police, already engaged in an investigation on base of the murders and newspaper articles, became active immediately and arrested C 4.

In anticipation that the fate of the gang now was decided, C

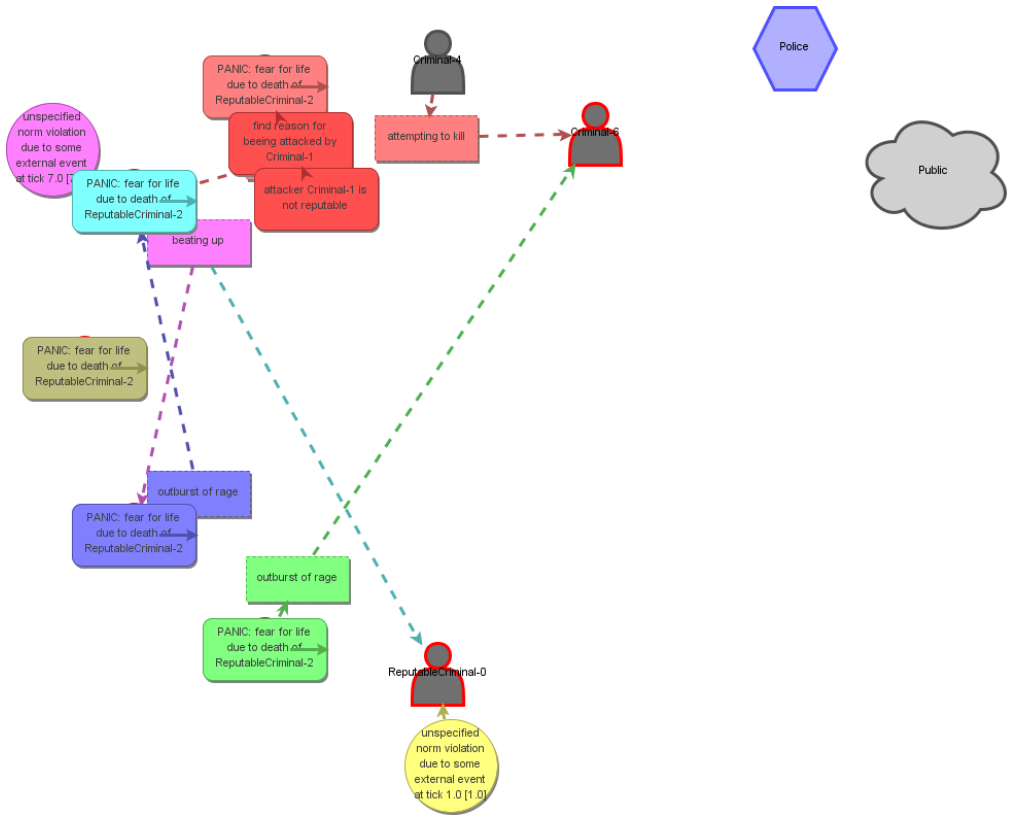

Figure 7.13: Simulation visualisation at tick 11 where a new series of violence started. The different colour express independent reactions, mostly caused by panic due to the observed murder of RC 2

> 0 and C 2 tried to screw out the money they had given WC for laundering, in order to escape to foreign countries. They started intimidating WC. This set off new acts of betrayal and violence, resulting in the death of C 0. Only when WC was arrested by the police, the story of the gang came to an end: Only C 2, RC 1, C4 and WC survived, the former two together in the underground, the latter two in prison (Figure 7.14).

The scenario above represents just one of the discovered patterns. With the basis model (that does not change the reputation during simulation runs), extensive experimentation and analysis has been done by Neumann and documented e.g. in (Lotzmann and Neumann, 2016). In these experiments the proportion of Black Collar and White Collar criminals was varied, and several dozens of simulation runs were evaluated, classified and finally compiled into model-based scenarios. In (Lotzmann and Neumann, 2016) the scenarios did "not represent the full behaviour space of the model but only those that are

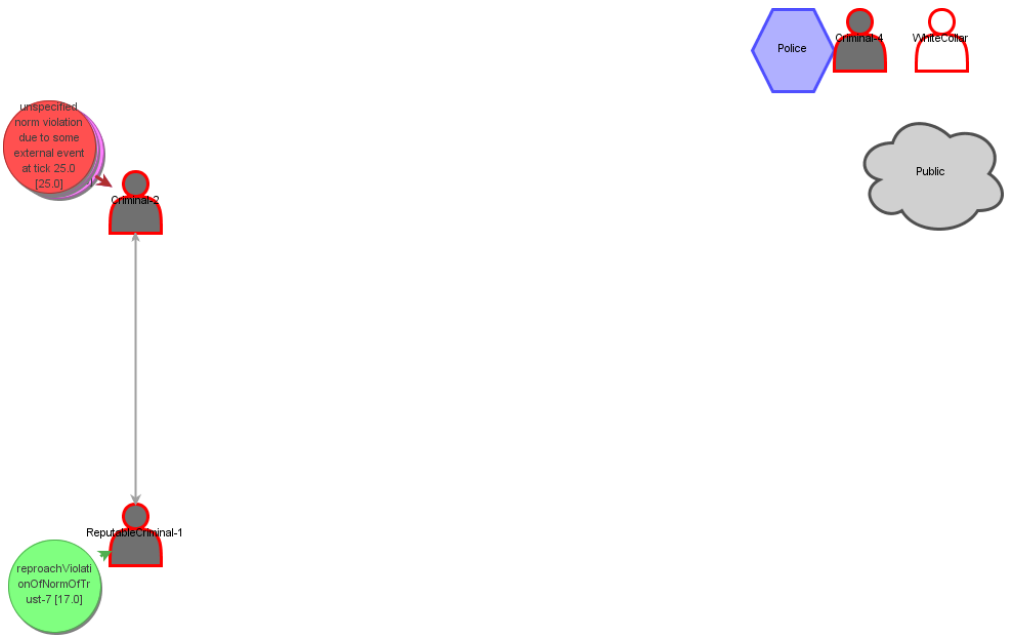

Figure 7.14: Simulation visualisation of the final state at tick 39

of interest for examining modes of conflict regulation and outbreak of violence in a group with properties comparable to the empirical case, namely a group with no managerial authority assigned to certain positions such as a 'boss' or 'godfather' in a professional, Mafia type organization". Given this context, four scenarios have been selected for presentation in (Lotzmann and Neumann, 2016):

The first scenario, "eroding of a criminal group by increasing violence", describes a similar path as exemplified above, though in a slower pace (i.e. involving more ticks) and with a more prominent intimidation (and overall role) of the White Collar criminal. Here, the initial event of an unspecified norm violation strikes an ordinary criminal (in contrast to the reputable criminal in the example above), but also in this case an initial bilateral conflict ends with an assassination, which sets off a panic. This can be seen as an example very close to the genuine events that shattered the real criminal network.

The second scenario shows a case where violence does not escalate, but just caused a "small irritation". In this case the initial norm violation is associated with the White Collar criminal, and the high reputation (and hence also initial image) as well as the capability to acquit the debt with money helps to prevent any violence.

The third scenario is dedicated to a pattern where "the group overcomes severe escalation of violence", again due to the ability and willingness of the White Collar to pay the debts. In this case, even a partly successful police

intervention (by arresting one of the gang members) could not stop the criminal business.

This is different in the fourth scenario, where "successful police operations" smash the criminal network by efficient strikes against key criminals, most importantly the White Collar.

The conclusions on observable "central mechanisms" drawn from these four scenarios can be summarised as follows:

- Bilateral conflicts between members of the criminal network may go on for long periods without affecting the group.
- Ambiguity of violence — the cause behind panic due to fear for own life — stimulates spreading of mistrust.
- The White Collar is the most vulnerable criminal in the network, but can break the cycle of violence by fulfilling requests for money by Black Collar criminals.
- Police operations are most effective against the White Collar criminal.
- If significant numbers of criminals change sides by betraying the criminal network, less bloodshed can be observed, mainly due to more effective police operations.

7.6 Conclusions on the use case model

After pointing to the methods and tools used for verification and validation of the use case model in the first part of this chapter, actual results from the simulations are presented. Thereby, the overall research question "How could this escalation of violence happen?" — as raised in section 5.2 — can be considered answered by unfolding the involved internal dynamics of the criminal network.

Regarding the other — more specific — research questions, the following assertions can be made:

> Stakeholder: "Is the evidence provided rich enough to inform formal modelling and simulation?"
>
> Simulation Modeller: Yes, as proven by successful validation of the simulation model.

> Stakeholder: "From a criminological viewpoint, can new insights and results beyond the already concluded 'classical' police investigation about the case be achieved from simulation?"
>
> Simulation Modeller: Yes, by virtual experiences, as discussed in (Lotzmann and Neumann, 2017) and (Van Putten and Neumann, 2018).

> Social Scientist: "Is it possible to gain insights on the fundamental social-theoretical question of conditions for social order from analyses on a microscopic (i.e. individual) level?"
>
> Simulation Modeller: Yes, the emergence of a normative authority and the conditions under which this happens can be examined with the simulations. More detailed insights to this question are given by the Data Analyst e.g. in (Neumann and Lotzmann, 2016a) or in (Lotzmann and Neumann, 2017), section "Interpreting simulation: Growing criminal culture".

The short answer to the research questions from the computer science perspective raised in section 5.2 is 'Yes', as the existence of this simulation model shows. A more elaborate discussion on these questions is provided in the final conclusions in Chapter 11.

Part III

Technical basis

Chapter 8

DRAMS: A declarative rule engine for agent-based simulation

8.1 Introduction

This chapter provides the 'entry point' to the second practical part of the thesis — DRAMS as the 'engine room' for the use case model presented in Part II. The chapters of this part cover aspects such as operating principles, software design, architecture and implementation, together with an outline of the development process. Here, mainly the discussion provided in section 2.2 sets the context.

This Chapter 8 provides the conceptual design and development process perspective. DRAMS is the result of cooperative design and development activities; details are given in section 8.4.1. Parts of the content have been disseminated before in (Lotzmann and Meyer, 2011a) and (Lotzmann and Meyer, 2011b), as well as in project deliverables mentioned below.

The chapter is structured as follows: After a motivation, a section on the design basics provides some theoretical background and the basic working mechanisms of DRAMS. This section is followed by a segment on the development lifecycle, covering insights on the development process of DRAMS and the applied testing and deployment methods. Finally, two different views on the DRAMS interfaces are characterised: for the implementer of DRAMS models and for the user of simulation models developed with DRAMS.

8.2 Motivation

Declarative programming languages have been used for several decades, in particular in the field of artificial intelligence (in the broadest sense) they have gained a certain degree of popularity (see section 2.2.6). Declarative rule engines — as an example for the practical realisation of declarative programming

languages — allow to 'solve problems' not by specifying an algorithm which specifies what the way to the solution looks like (like in the imperative programming paradigm), but rather are based on logic calculi specifying what the solution itself looks like, without taking the way to the solution into account. The most popular logic-based programming language Prolog (Kowalski, 1988), developed in the 1970th, is an example of the branch of problem-solving systems based on a mechanism called backward chaining, which tries to deduce the cause(s) of existing facts. A different (or rather opposite) approach is called forward chaining, where possible outcomes are inferred on base of given facts, as it is done in expert systems (Leondes, 2002). Such systems are referred to as production systems, or more generally[95] declarative rule engines. A contemporary system of this sort is JESS (Friedman-Hill, 2003), with popular ancestors like OPS5 (Forgy, 1981). This approach is also implemented in DRAMS, where a rule-based inference mechanism is applied to agent based simulation, i.e. to model agent reasoning in a kind of expert system.

This chapter illustrates a path followed in the OCOPOMO project, where DRAMS as a new rule engine software was developed (see section 3.3.4), distinguishing it from other available systems (some of which already have been applied for agent-based simulation) mainly by aspects particularly relevant for evidence-based multi-agent simulation, like distribution of reasoning among agents, transparency of rule processing and traceability of model artefacts to evidence. These aspects are introduced and illustrated in the first two parts of this PhD thesis and constitute the special requirements for DRAMS.

8.3 Design Basics

In the following two sections an introduction to DRAMS is given. Firstly, the theoretical grounds are outlined, in order to make the following description of the DRAMS operating principles comprehensible.

8.3.1 Theoretical background

Artificial intelligence of software agents in social simulation is often represented by rules — besides other approaches like artificial neural networks (see section 2.3.1). While there are many different ways to specify and process these rules, most approaches have in common that agents are equipped with sets of rules, where rule processing is triggered by events occurring within some kind of artificial world which are perceived one way or other by the agent. Such events can for example indicate that some period of time has elapsed, that an obstacle appeared in an agent's direction of movement, or just expressed more abstractly: a new fact has been added to some data base. Such facts do

[95] Rule Engine refers to a more general concept, and can either be imperative or declarative, while the latter can include both forward and backward chaining characteristics.

not necessarily represent real 'facts' (in the sense of 'law of nature'), but often rather beliefs, i.e. an agent's persuasion about a fact. The result of a rule execution can then be an action, changing the state of the environment. Such rules can be represented for example by stochastic decision trees (as used e.g. for normative simulations in EMIL-S; Conte et al. (2013)), or by deterministic condition-action rules (like e.g. `if some_event then do some_action1`), which can also take the state of the agent into account (e.g. `if some_event and (some_variable equals some_state) then do some_action`). In the latter case, rule processing can be done by simple production systems (Gilbert and Troitzsch, 2005), but for models exceeding a certain degree of complexity rule engines with sophisticated supportive features are indicated, for example with respect to a full-fletched rule specification and programming language or a tight integration within a tool-supported modelling process (like the OCOPOMO process introduced in Chapter 3). A few examples where such declarative rule engine technology has been successfully applied in the field of agent-based modelling are:

- The Smalltalk-based tool SDML (Strictly Declarative Modelling Language, Moss et al. (1998)) has been used for water demand management models within the EU FP5 project FIRMA (Freshwater Integrated Resource Management with Agents, Barthelemy et al. (2001)).

- A model of the social impacts of HIV/AIDS in villages of a rural province in South Africa was created in the context of the FP6 project CAVES (Complexity, Agents, Volatility, Evidence and Scale; Alam et al. (2007); Moss et al. (1998)) using the Java-based rule engine JESS[96] (Friedman-Hill, 2003).

- In the OCOPOMO project, several policy models have been developed with DRAMS, for example to explore different energy policy options in the self-governing region of Kosice (Slovakia) and a model on knowledge transfer strategies for improving industrial development in the Campania region (Italy)(Bicking et al., 2013).

Declarative rule engines combine production rule inference with the paradigm of declarative programming. Declarative programming languages are related to mathematical notations, which are quite different to typical imperative programming languages based on control structures, such as Java or C++ (see section 2.2.6). Writing (or understanding) programs in these two paradigms require different approaches and ways of thinking. Hence, for (e.g.) experienced Java programmers it can become troublesome to create good[97] declarative programs, and the other way around. This is where graphical notations

[96] `http://www.jessrules.com`

[97] 'Good' in a sense of a true declarative approach, not just trying to imitate imperative control mechanism in declarative programs.

become helpful which can provide a more 'intuitive' perspective on declarative code, like the DDGs introduced in Chapter 6. Such representations show an interesting property of the declarative specifications: a contiguity to natural language. This makes it easier to express natural language descriptions into rules, and on the other hand to generate (sort of) natural language as an outcome of programs (story telling machine, as exemplified in Chapter 7).

The approach followed with DRAMS is not a pure declarative one, since it gives the developer control over temporal dependencies instead of implementing an 'intelligent' conflict resolution mechanism. A consequence of this design decision is that efficient DRAMS rules should consider temporal relations between rules, a requirement that (in the author's experience) helps to structure behavioural agent rules and therefore represents a reasonable constraint.

From an application perspective presented in the previous chapters of this PhD thesis, a forward-chaining approach is suitable to reflect the necessary capabilities of the agents. However, backward-chaining capabilities of a rule engine might add sophisticated reasoning capabilities to agents, if required by a particular model (like interpreting other agents' actions), so that this is a possible field of future extensions to DRAMS.

As mentioned e.g. in (Lotzmann and Meyer, 2011b), the general concept of a forward-chaining rule engine is understood as a software system that basically consists of the following components:

- A fact base, which stores information about the state of the world in the form of facts. A fact contains a number of definable data slots and some administrative information (time of creation, entity that created the fact, durability).
- A rule base, which stores rules describing how to process certain facts stored in fact bases. A rule consists of a condition part (called left-hand side, LHS) and an action part (called right-hand side, RHS).
- An inference engine, which controls the inference process by selecting and processing the rules which can fire on the basis of certain conditions. This can be done in a forward-chaining manner (i.e. trying to draw conclusions from a given fact constellation) or backward-chaining manner (i.e. trying to find the facts causing a given result).

These components are characterised and detailed below from a conceptual point of view, and in the following chapter from a more technical perspective.

8.3.2 Operating principle of DRAMS

The development of DRAMS was mainly driven by the following requirements:

- Create a declarative rule engine that is optimised for changing fact and rule bases, without impairing the performance of the system, even with

- large and distributed fact bases. Distributed fact bases were taken into account to cater specifically for the envisaged agent concept, i.e. fully autonomous entities (according to (Gilbert and Troitzsch, 2005)) with self-contained rule engines.

- Providing an experimentation platform tailored for the design decision to implement a data-driven approach, i.e. to try different functionalities and to be able to monitor the rule engine processes. This point also refers to

- the most important requirement: the preservation of the evidence traces from the evidence base, via the conceptual model to the simulation results. This requires also an

- integration of the rule engine into a toolbox supporting all phases of modelling and simulation.

DRAMS is designed as a distributed, forward-chaining rule engine. It equips an arbitrary number of agent types with type-specific rule bases and initial fact base configurations. For each agent type, an arbitrary number of agent instances (objects) with individual fact bases can be created. All individual fact bases are initialized according to the agent type configuration, but may be adapted individually. There is also a shared global fact base, containing 'world facts', e.g.

- a (permanently updated) fact reflecting the current simulation time,

- one fact for each agent instance present in the 'world', providing some information (e.g. reference ID) about the agent,

- model-specific environmental data, and

- (public) inter-agent communication messages.

Heart of the inference engine is the rule schedule, an algorithm deciding which rules to evaluate and fire at each point of time. The following pseudo code shows the basic structure of a possible algorithm:

```
processSchedule(time t){
   while new facts are available at time t
   loop

      find all agent instances for which new facts are available;

      foreach agent instance
      loop
         find all rules for which new facts are available at time t;
```

```
         foreach rule
         loop
            evaluate LHS;
            if evaluation result==true then
               execute RHS;
               // e.g. generate new facts
            end if;
         end loop;
      end loop;
   end loop;
}
```

In order to decide (for each fact base configuration without recompiling the rule base) which rules to evaluate for which agent instances, the scheduler relies on a data-rule dependency graph. This is constructed once from all specified rules and initially available data; the graph does not change unless rule bases are modified. As to detecting fact base modifications, the schedule keeps track of all (writing) fact base operations.

Figure 8.1 shows a stylised example of such a data-rule dependency graph, containing five rules working on four facts with different operations — retrieve and query for reading access, and assert for writing. At time *t*, Fact 1 and Fact 3 are available, so that Rule b and Rule f can fire in *schedule task 0*. Rule b, however, asserts Fact 2, which then allows Rule c and Rule d to fire in *schedule task 1*. The same applies for Rule f, and so on. The resulting complete evaluation tree is shown in Figure 8.2.

At each point of time, the rule processing within an agent (intra-agent process) is performed for all possible rules. At first, the conditions of a rule are checked, i.e. the LHS is evaluated. Each LHS consists of one or many (LHS) clauses, pertaining to the following basic categories:

- Clauses for retrieving data from fact bases. These operations are similar to data base queries, where as a result a set of facts (with 0, 1 or many elements) is retrieved. Each retrieved fact is called an instance of this clause, and all subsequent clauses of the LHS have to be evaluated for all instances. Thus, this clause type is spanning an evaluation tree. If one or more facts are retrieved, the evaluation result for this clause is true, otherwise false. In the latter case, the evaluation of this tree branch is terminated.

- Clauses which test whether data from the retrieved facts hold for specified conditions. If such a test fails, the evaluation of this tree branch is terminated. The set of leaves of the evaluation tree is considered a set of possible input data configurations for firing the RHS of the rule. The RHS consists of one or many (RHS) clauses with the purpose of executing fact base operations (adding or removing a fact[98]) or other actions (e.g.

[98] An operation for changing a fact is not implemented — see further explanations below.

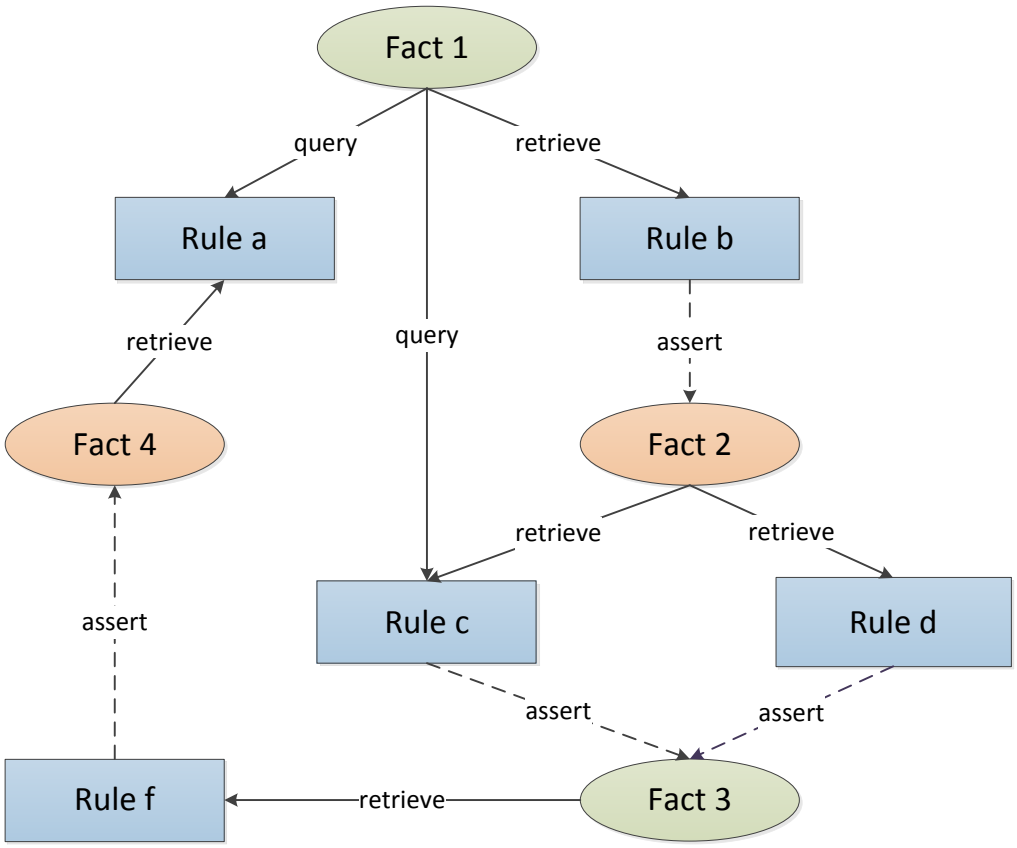

Figure 8.1: Data-rule-dependency graph (configuration at time t; legend: green ellipse — existing fact, red ellipse — fact asserted by a rule, blue rectangle — rule, solid arrow — reading fact base operations, dashed arrow — writing fact base operation)

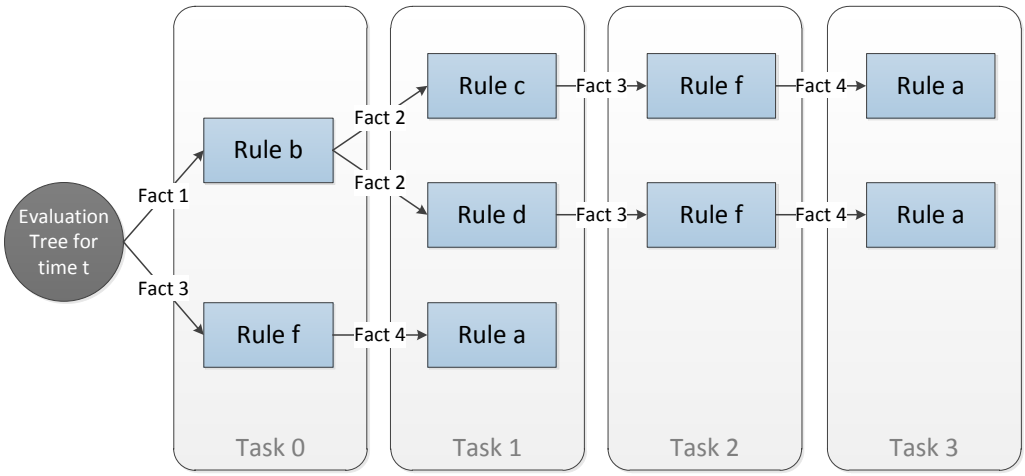

Figure 8.2: Evaluation tree for time step t

printing a statement to a log).

Accordingly, the expressiveness of the system is determined mainly by the number and capabilities of available clause types. In DRAMS, the following functionality is available for the LHS of a rule:

- Data from fact bases can be obtained either by retrieve or by query clauses. In both cases, a query on a fact base is performed, and a number of facts, for which the slot values specified in the query match and optional time-related conditions hold, are returned. In the case of a retrieve clause, these facts are used to create a corresponding number of instantiations, and for each instance a number of requested variables are bound with specified slots. In the case of a query clause, only one instance is created, and a list of retrieved facts is bound to a result variable. These two clause types are representative for the first category defined above, whereas the following LHS clause types belong to the second category.

- A unary BIND operator, which binds a specified variable with the evaluation result of an expression. The expression can be another single (already bound) variable, or a more complex arithmetic expression (with a standard repertoire of math operations, including generation of random numbers).

- A full set of binary logical operators. The result is either bound to another variable or, alternatively, used as clause evaluation result.

- A set of LIST operators, including generation and modification of lists, counting and extracting of list elements, and several accumulator operations (sum, avg, min, max etc.).

- A set of SET operators, including creation and modification of sets, number or existence of elements, as well as union and intersection of two sets.
- A NOT clause, inverting the evaluation result of the specified inner clause. The inner clause can be any other LHS clause.
- A COMPOSITE clause, which can be seen as an encapsulated LHS with its own variable name space. For processing the specified inner clauses, the evaluation mode can be chosen between AND, OR and XOR.

RHS clauses dedicated to perform actions comprise:

- Asserting new facts to (in principle any existing) fact bases.
- With some restrictions[99], retracting existing facts from (in principle any existing) fact bases.

Facts are associated with a time when they are valid (or true). This implies that a fact cannot simply be changed, when at a point of time the information carried in the fact is no longer true. Instead, the fact has to be overridden (or possibly retracted and replaced) by a new one, which is then valid from the time of assertion on.

There is one special clause defined which can be part of both the LHS and RHS, providing two kinds of actions:

- printing formatted text (including values of variables) to a log (either to a console window or to a file);
- calling a method on the underlying model part; the peculiarity of this functionality is explained in more detail in the following section.

An important aspect of any multi-agent simulation system concerns the means by which agents can communicate with each other (inter-agent process). Technically, DRAMS provides three options:

- communication via the global fact base in a blackboard-like manner;
- writing facts to fact bases of other (remote) agents; this can be interpreted as the way humans typically communicate with each other, via speech or written messages;
- reading facts from fact bases of remote agents; this is conceptually similar to mind-reading, and should thus be avoided in most cases, but can be useful to find out (public) properties of another agent, or could be exploited e.g. by external observers.

[99] If the past doesn't matter, old facts can be retracted. Otherwise, facts can be overridden by new facts valid from the time of assertion on.

DRAMS implements two operation modes for `RuleSchedule`. The active time mode (Figure 8.3) is used for time-driven simulation runs, where the time steps are set by a simulation scheduler (which controls the flow of simulation time, not to be confused with the rule schedule). The passive time mode (Figure 8.4) provides means for event-driven simulations. No external time step is fed in by the simulation controller, instead the rule schedule proceeds with the next point of time at which some system state change is scheduled (e.g. indicated by deferred fact assertions). When using passive time mode, complete simulation models might be specified and run solely with the functionality provided by DRAMS. However, if some kind of environment beyond fact base entries is to be included into a model designed for discrete time step operation, DRAMS only takes care of agent deliberation abilities, while the environment is outsourced to an external simulation tool. Since DRAMS is implemented in Java, in principle any Java-based simulation tool is qualified for this purpose. As detailed in section 8.4.3, DRAMS can be used as a rule engine extension for Repast (North et al. 2006). In this case, a Repast model class maintains a link to the DRAMS rule schedule, and each Repast agent instance maintains its own rule engine object. In order to guarantee seamless integration, DRAMS provides several means for accessing Java objects (e.g. shadow facts, Java action clauses). All these features are applied in the model described in Chapter 6.

8.4 Development Lifecycle

The following three subsections cover aspects of the development of DRAMS. Firstly, the development approach, the actual process and the time-line are outlined, followed by reflections on the incorporated testing measures. Finally, the deployment subsection provides information on distribution and some technical details on a concrete usage scenario of DRAMS in agent-based simulation models.

8.4.1 Development Process

The development of DRAMS was done as part of the OCOPOMO project as a collaborative effort of several partners. The general design idea and initial requirements were provided by SMA[100], the technical design and implementation was done by UKL[101] and (in the first stages of development) MMU[102]. The process of developing DRAMS was inspired by the design of SDML, a strictly

[100] Scott Moss Associates (SMA), Chapel-en-le-Frith, U.K. — who also contributed the acronym DRAMS

[101] Research group eGovernment, University of Koblenz-Landau (UKL), Germany; performed by the author

[102] Centre for Policy Modelling (CPM), Manchester Metropolitan University (MMU), U.K.

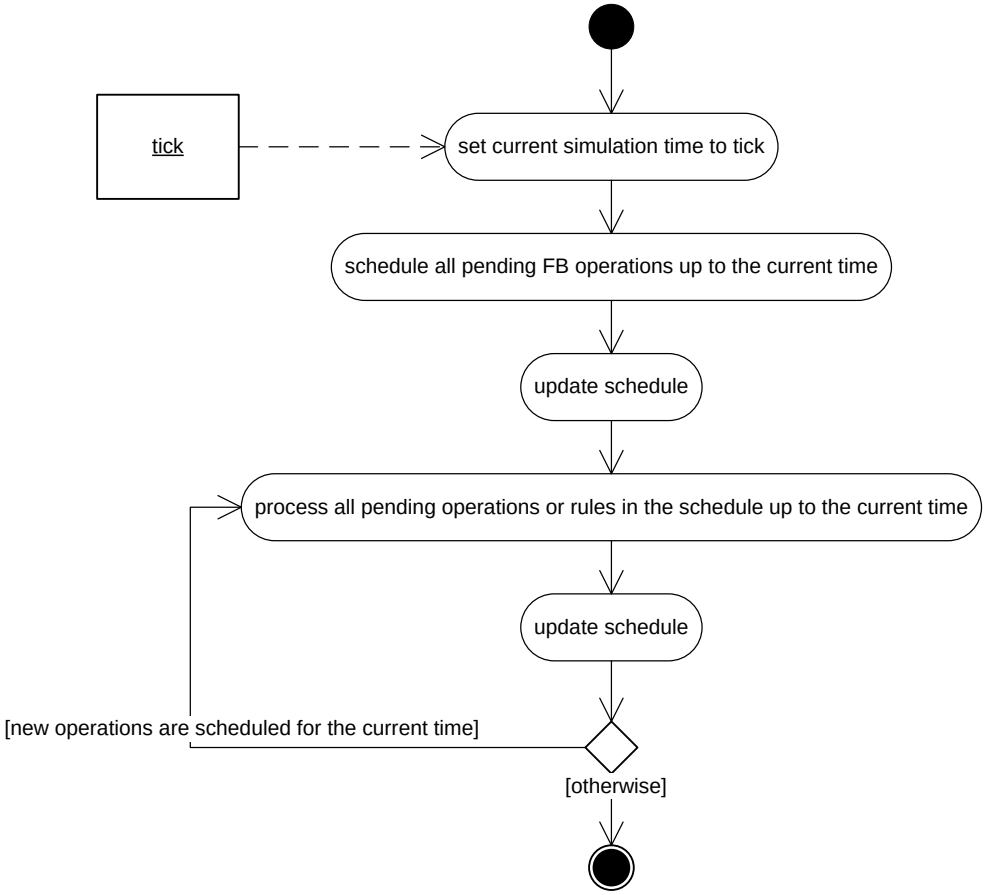

Figure 8.3: Active time (round based) activity diagram for processNextTick(tick)

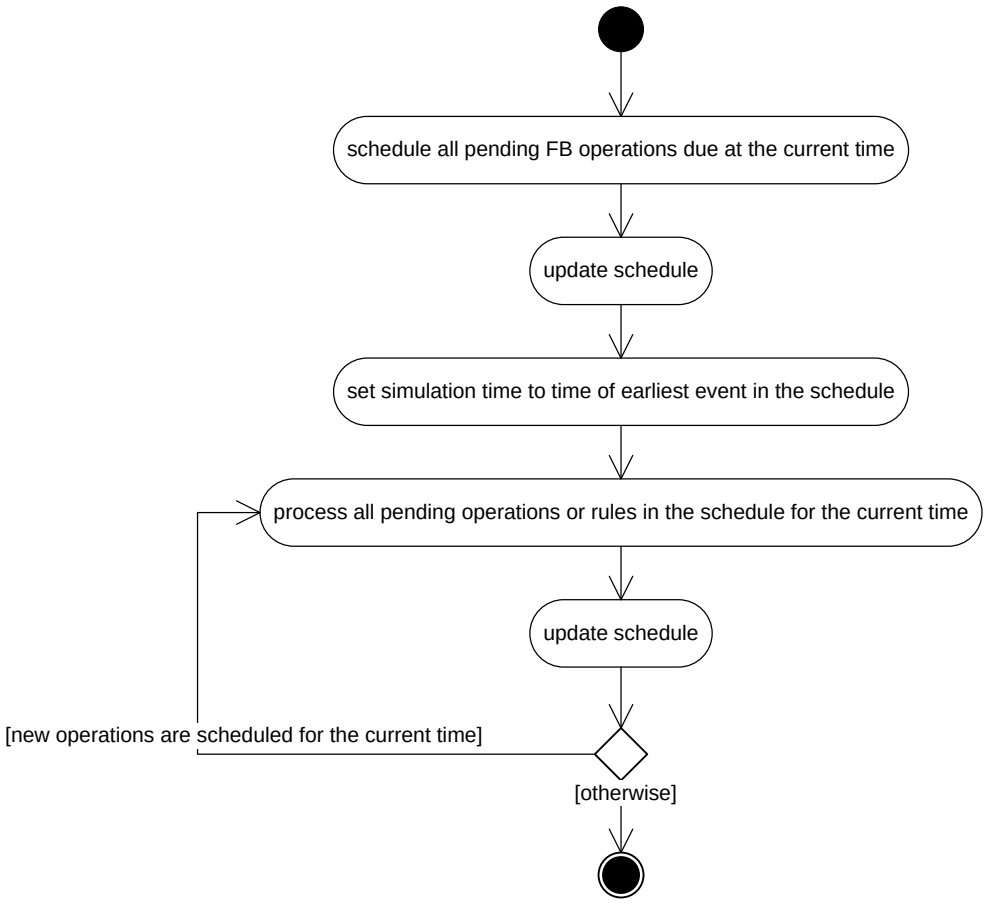

Figure 8.4: Passive time (discrete event) activity diagram for processNextTick()

declarative agent modelling language (Moss et al., 1998). The decision to create a new rule engine was fostered by the facts that maintenance and further development of SDML was ceased and could not be used in the available state of maturation, and that the aforementioned special requirements (in particular traceability) could not be satisfied with other readily available rule engines at that time. The two Java-based rule engines JEOPS (Filho and Ramalho, 2000) and JESS (Friedman-Hill, 2003) had been examined, where additional complications were experienced. These complications were — shortly speaking — related to fact bases changes which were described as causing frequent recompilation of the networks used within the RETE algorithm (which most likely occurred due to excessive working memory usage), hence inflicting severe performance issues. The point of departure for DRAMS was the assumption that a naive data-driven approach could overcome the burden of such frequently repeated and time-consuming recompilations at the price of an overall less efficient rule processing.

Several factors led to applying a rapid incremental prototyping process:

- The initial requirements mentioned in section 8.3.2, together with
- the high likelihood of changing and emerging requirements,
- the project environment with restrictions on available resources (time and personnel), and
- the preferences of the involved partners.

This can be seen as a contrasting approach to the example model development detailed in Part II of this PhD thesis, where conceptual modelling and specifications set the base for the actual implementation. These two software development approaches will be taken up again and compared to each other in the final Chapter 11.

The rapid development of the DRAMS prototype was strongly associated with the acquisition of new requirements, both functional (as the shape of the system became clearer) and non-functional, as the usage of DRAMS revealed issues. The story of developing DRAMS will be outlined in the following, highlighting the most relevant requirements. The full development history is provided in (Lotzmann, 2013).

The development of DRAMS started in May 2010 in what could be called a first sprint[103] in which the fact and fact base infrastructure with query functionality and the basic rule base and rule components were implemented. At this time rules could be written that performed simple fact base (retrieve) and compare operations.

This successful 'proof of concept' associated with this stage of development triggered ample extensions of the fact base and rule components in the following months. In June 2010 generalised rule evaluation code was written, logical

[103]The term sprint is used in the meaning as defined by the Scrum development method as defined in section 2.2.1.

operators and writing fact base (assert) operations added, the first version of the data-dependency graph (DDG, together with a visualisation in a graphical user interface) took shape.

In July 2010 the distribution of rule and fact bases among agents was introduced, making it possible to manage and visualise distributed dependency graphs, i.e. graphs incorporating just the individual fact and rule bases of agents and the newly incorporated global fact base (providing among others a communication blackboard for the agent rule engines). These functionalities (in particular the global fact base) demanded the implementation of a global 'rule engine manager', as well as the possibility to access not only the individual fact base from within rules, but also remote fact bases (i.e. the global fact base, but also other agents' fact bases, enabling one-to-one communication among agents). As central part of the rule engine manager the rule schedule concept was developed as the core of the inference engine, to determine the rules to fire at any specified point of time. At this time another requirement became desirable: a way do define valid structures and data types stored in facts. For this purpose fact templates were introduced and implemented in the fact bases. Furthermore, a plethora of functionality for rules was implemented in form of clauses, including temporal constraints on fact base operations (e.g. deferred access to facts), mathematical expressions, list operators and a clause to invoke functions provided by Java code.

While implementing experimental simulation models with the functionality of DRAMS available at this stage, new functional and non-functional requirements became apparent, which were tackled in August 2010. The data-driven rule scheduler was optimised, (deferred) fact base operations improved for performance and stability reasons. Several new clause types for fact base queries and accumulative list operations were added, and a new fact type was introduced — a more versatile shadow fact (for data exchange between declarative rules and model parts written in Java). Furthermore, the graphical user interface was featured with a display for the rule schedule.

In September 2010 the basic functionality of the rule engine was present. In a second sprint until April 2011 the prototype was matured and equipped with additional functionality in several iterations. As the main functional requirements could be regarded as complete, the final architecture was designed and the respective class structure was created in a refactoring process. Until then the facts and rules had to be created by Java object instantiations, which was complemented by a parser component. For the parser implementation a loose coupling approach was applied, meaning that the DRAMS core implementation remained untouched, while the parser just collaborated with object factories for creating objects for facts, rules etc.. The syntax of the parser was designed on base of the OPS5 syntax.

A third sprint followed until June 2011, mainly coping with stabilisation and bug fixing, but also adaptations caused by small requirement changes. These included changes in the rule scheduler code to support new lag modes

(i.e. ways to specify temporal constraints on fact base operations) and fact retractions. The global fact base was extended by rule processing functionality, the data types in fact template definitions could be specified by enumerations of allowed values. A few declarative language constructs were perceived as clumsy, leading to slight modifications of the parser syntax. Finally, some corrections and additions to the graphical user interface and the interface to RepastJ model code were implemented.

A fourth sprint was done until October 2011 with similar intentions of bug fixing, performance and stability improvement and new user interface functions, such as more expressive error messages and DDG visualisations with added information.

A fifth sprint until January 2012 brought several important features:

- The traceability infrastructure by generating and storing an evaluation tree.
- An on-the-fly rule processing feature which required changes both in the DRAMS core as in the graphical user interface.
- An integrated profiler to measure the rule processing time.

After these five sprints from time to time new requirements for new functionality and bug reports came in, related to realisations of new DRAMS applications, i.e. modelling projects that used DRAMS as technical basis. These were incorporated into DRAMS until April 2013 and included changes like additional clauses, agent 'birth' and 'death' functionality, and the finalisation of the traceability functionality. A refactoring sprint in October 2012 introduced a plugin interface, removing all GUI code from DRAMS core and at the same time creating respective plugin modules.

By 2017 the code is still maintained, modernised and equipped with new functions, as required by modelling projects. So have extensions to the simulation model introduced in Chapter 6 led to the implementation of new fact base retrieve options (new lag modes).

The different components and functions are elaborated in the following sections from design and usage perspective, and in the next Chapter 9 architecture and implementation aspects of DRAMS are discussed.

8.4.2 Testing DRAMS

Unlike typical extreme programming methods that rely on formal unit tests as integral part of the software development, testing during the implementation of DRAMS was API oriented by design. This means that the different modules of DRAMS are tested with appropriate test rules specified in a 'simulation model' project with the purpose to verify the different modules of DRAMS. This project is designed in a similar way to test classes, i.e. the execution of this model performs the API functions (i.e. the different clauses) and results

in an 'OK' message in case of success, or the source of a failure otherwise. The test model extensively covers the fact base operations with all the available operators and lag modes. Other modules (with functionality like accumulators, logical, list and set operators) are covered on a basic level, so that an API test set for all core functions is available.

Reasons for favouring this approach over the classical unit tests were basically not only the initially blurry (or even unknown) structure of the software system and the requirements for the modules, but also the initially blurry API functionality itself. The requirements of the API were concretised step by step by the involved modelling experts, so that the verification of API was crucial and — with the assumption that correct API functionality implies correct module functionality — sufficient.

The basic structure of this test model follows a typical simulation model. Besides the main model declarative definition (global.drams) file with a global evaluation rule that collects the results of all the module tests and prints the final result, the single modules are tested by dedicated 'agents'. I.e., all clauses related to a DRAMS module are tested with rules assigned to the related agent. The most important and complex agents `TestActorRetrieve` and `TestActorQuery` are testing the fact base operations. Other actors like for example `TestActorList`, `TestActorAccumulator` and `TestActorOperators` deal with list, list accumulator and (logical and other) operators, respectively.

Each agent is performing the test for its module during a number of simulation steps (12) and reports logs and errors to an output XML file (and to the console), and writes a success fact to the global fact base. As soon as all expected success facts are available, the global evaluation rule reports success and stops the execution of the model.

The test model is available online, see appendix C.3.

8.4.3 Deployment

DRAMS is provided as Open Source Software and distributed as a Java Archive. It is not intended to be used as a stand-alone Java application, but as a software framework together with Java applications in general, in particular with Java-based simulation tools. Ways to obtain the software and installation hints are mentioned in appendix C.2.

Figure 8.5 shows a typical usage scenario of DRAMS together with the RepastJ (North et al., 2006) simulation software as a class diagram, pointing out the relevant components and classes, and the dependencies between them. The DRAMS package consists of a general component (called DRAMS in the figure) and a platform specific component (named DRAMS Platform), here specifically for RepastJ. The latter contains a `Model` class, which is a subclass of `SimModelImpl` of RepastJ, and which implements the `IModel` interface of DRAMS, bringing functionality of both frameworks together. The link from the `Model` to the DRAMS functionality is expressed by the associations to the

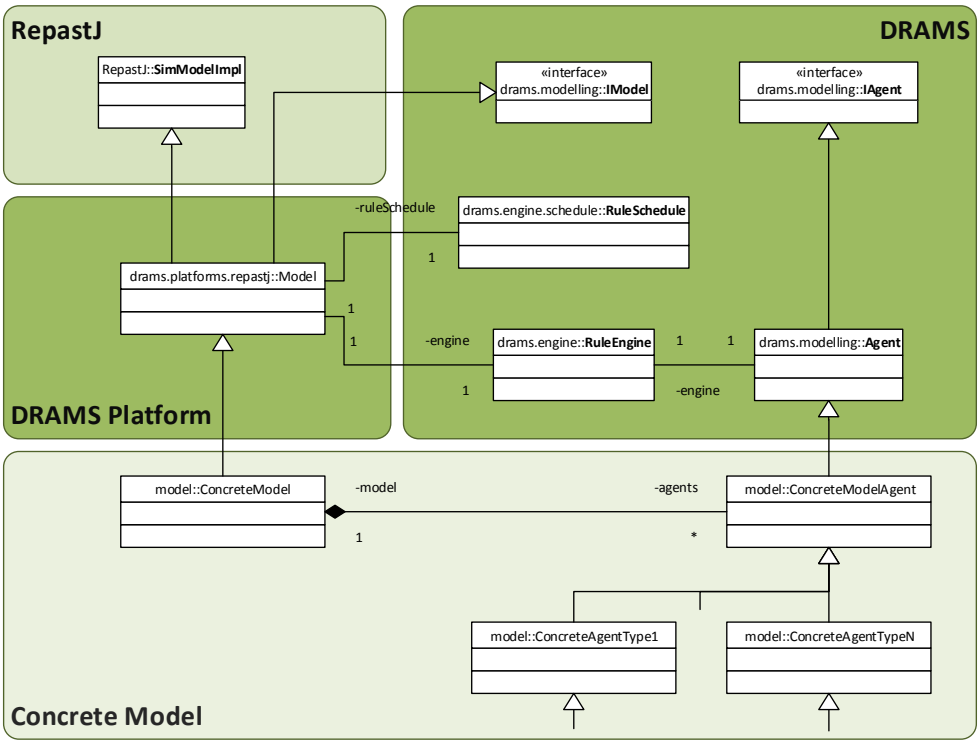

Figure 8.5: DRAMS model class diagram

`RuleEngine` and `RuleSchedule` classes. The `RuleEngine` associated to the `Model` contains the world facts and global rules, while with the `RuleSchedule` the simulation model controls the rule engine functionality. Also part of the DRAMS component are the `IAgent` interface and a general `Agent` implementation, which is also associated with `RuleEngine` for agent-specific facts and rules.

A concrete simulation model has to foresee a `ConcreteModel` implementation of platform `Model` class, and typically a `ConcreteModelAgent` class as a general agent concept associated with the concrete model class. All agent types needed in the model are then implemented by several different `ConcreteAgentType` classes. The model facts and behaviour is incorporated in the `ConcreteModel` and the `ConcreteAgentType` classes.

The entire process of implementing and running a simulation model based on this deployment scheme can be structured into five steps:

1. A Repast model class must be defined, which creates all agent instances (for which the desired agent types have to be defined, see step 2), initialises the global fact base and handles the Repast time steps by triggering the rule scheduler. An abstract model super class providing the

DRAMS related code is available with the platform-specific `Model`, thus only model related aspects have to be added.

2. Classes for all designated agent types must be created. Besides environment-related functionality, code must be prepared for initialising the rule engine and for defining the agent (type and instance) specific fact templates, facts and rules. Similar to the model super class, an appropriate `Agent` super class is delivered with DRAMS.

3. For each agent class, code for the declarative model part has to be written. Firstly, the fact templates have to be specified, after that the initial facts can be asserted to the fact base, and finally the rules can be written. In general, all DRAMS related elements can be either specified by directly instantiating the appropriate Java objects, or by using the DRAMS modelling language (in this case the Java objects are created by the DRAMS parser).

4. The declarative model part can be developed and tested with the DRAMS user interface (e.g. checking for consistency using a visualisation of the automatically generated dependency graphs).

5. The model can be executed using the Repast user interface.

The user interfaces mentioned in these steps are detailed in the subsequent section.

8.5 Interface design

For any simulation system, typically two types of use cases can be distinguished which are associated with different perspectives on the user interface:

- a modelling perspective, designed to the purpose of writing and testing model code, and

- an experimentation perspective, allowing the user to configure, control and analyse simulation runs.

Figure 8.6 shows these two user interfaces as front-ends for a simulation model based on DRAMS and RepastJ. In the following, a brief overview on the relevant relations between front-end and simulation model are given. More detailed information can be found in (Lotzmann, 2013) and the DRAMS documentation (appendix C.1).

The dashed arrows in Figure 8.6 reflect the module dependencies between RepastJ and DRAMS. The classes shown in Figure 8.5 are also reflected here by the Model, Agent, RuleEngine and RuleSchdule components. In addition components that are relevant for user interfaces are included: RuleBase and

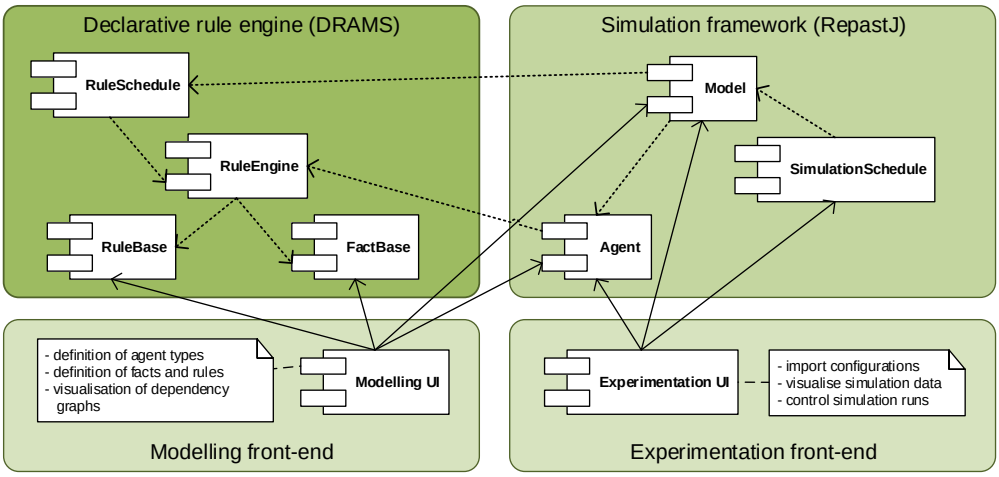

Figure 8.6: User interface components for DRAMS-based simulation models

FactBase as essential collaborators of RuleEngine, and the Repast SimulationSchedule that controls the simulation flow. The solid arrows show the dependencies between these (back-end) components and the two user interface front-ends elaborated in the following subsections.

8.5.1 Modelling user interface

The main functionality of the modelling user interface is to support the creation of the model structure and the coding of the different parts of the model. In this case, this is basically Java-code for the initialisation of the model component, the definition of agent types and related Java-code within the Agent component, together with the definition of rules and facts for the related RuleBase and FactBase components, respectively. Thus, the modelling user interface has to provide adequate code editors (for Java and declarative code) and supporting features like for example dependency graph visualisations, fact base inspectors and debugging tools.

Parts of the aforementioned requirements are fulfilled by readily available Integrated Development Environments (IDE) which in many cases allow to be customised and extended with additional features for specific needs. A popular example of a free open source IDE is the Eclipse[104]) framework and IDE, which has been selected as basis for the DRAMS modelling UI. So is the built-in Java editor used for editing the Java-code, and an Eclipse feature[105]

[104] `http://www.eclipse.org/eclipse/`

[105] This DRAMS editor feature has been developed together with DRAMS and can be installed from the Eclipse update site at `https://userpages.uni-koblenz.de/~ocopomo/release/` (login credentials can be found in Furdik et al. (2013)).

provides the editing functionality for declarative code (see screenshot in 3.9 in section 3.3.3).

The declarative code for DRAMS consists of four different top-level constructs, as introduced before in Chapter 3:

- Fact template, defining the structure of facts to be asserted to a fact base, including slot names and associated data types;
- Type definitions, specifying enum-like sets of strings allowed as values for slots of this type;
- Fact, an instance of a defined fact template.
- Rule, defines a set of actions to be performed if a specified set of conditions holds.

All these elements basically have to be inserted into the data structures of DRAMS in order to configure the rule engine for the intended purpose. The first three constructs have effect on the (agents' and global) fact bases, while the last one affects the rule bases. There are fundamentally two different ways to configure the DRAMS fact and rule bases: The first possibility is to instantiate objects of the respective DRAMS Java classes. So are — for example — `FactTemplate` and `Fact` classes available, from which objects can be instantiated. These objects can then be added to a fact base with the provided API methods. Correspondingly, instances of the Rule class can be created and added to a rule base. In this case the rule has to be further specified by adding clause instances to the rule object for condition and action parts.

The second possibility is to create these objects not manually, but by a provided parser. As stated above, the parser bundled with DRAMS relies on an OPS5-like language, and the above-mentioned Eclipse editor feature supports this language. But there are many other possible options: It is also conceivable to create a fully graphical user interface, allowing to configure DRAMS with a graphical notation like the Data Dependency Graphs introduced in section 6.3 — as an outlook for future work.

A thorough guide on the DRAMS language can be found in appendix C.1 and (an older version) in (Lotzmann and Meyer, 2013).

8.5.2 Experimentation user interface

The experimentation user interface is contributed by the simulation framework, in this case RepastJ. This interface provides mainly functionality to control (initialise, start, stop) simulation runs, but also means for adjusting parameters. Both can be either done interactively via a graphical user interface, or by configuration files, allowing to perform automated batch runs with specific parameter settings. To monitor (interactive) simulation runs, Repast offers

means to display aspects of the simulation state, for example by printing log messages or by drawing time line diagrams for selected simulation variables. In addition, arbitrary ways to visualise a simulation can be added by Java-code — see example in Figure 7.13.

Beyond these simulation framework related capabilities, DRAMS offers further possibilities to supervise simulation runs and results. For example, the modelling user interface can be used during experimentation, which is particularly useful in the phase of model verification and debugging: DRAMS provides internal log information like a 'live' rule schedule or error events, and the fact base content can be inspected for each simulation tick. Also the on-the-fly rule editor and processing facility can be used (in an 'alienated' way) to produce specific model output, hence live code instrumentation can be achieved. These related UI elements have been described and illustrated with screenshots in section 7.3.

DRAMS brings also UI plugins specifically designed for experimentation: several configurable log windows providing (text) output generated by rules with meta-information, using distinct font styles to highlight the different kinds of information (see Figure 7.1 for an example).

8.6 Conclusions

The declarative rule engine DRAMS presented in this chapter is designed to fulfil the most important requirements of a rule engine extension for agent-based social simulation. It brings several novel aspects and achievements :

- To the author's best knowledge, it can be considered the only available distributed rule engine architecture dedicated for agent-based simulation.
- It employs a data-driven scheduling algorithm, relying on data-rule dependency graphs, that can be transformed directly into visualisations of declarative programs (the DDGs).
- Changes within rule bases (e.g. triggered by meta-rules) can be realised, but are not yet implemented.
- The transparent rule processing opens the opportunity to be used as an experimentation platform, with possible extensions of any kind (e.g. integrating with existing systems, adding additional languages etc.).

Chapter 9

Architecture and realisation of DRAMS

9.1 Introduction

This chapter provides a dissection of the technical architecture of DRAMS, with additional information on the realisation of some core concepts and the used frameworks. Apart from the system documentation provided with DRAMS (as a project deliverable), covering some related aspects, the content of this chapter is unpublished original work.

The chapter is structured as follows: Firstly, an overview on the architecture of DRAMS is given. In the following segment the DRAMS core components are characterised one after another — rule engine, fact and rule bases, scheduler, parser and components related to traceability. Subsequently, the DRAMS API is specified.

9.2 DRAMS Architecture Overview

To recapitulate, main requirements for the development of DRAMS include:

- Create a declarative rule engine that is optimised for changing fact and rule bases, without impairing the performance of the system, even with
- large and distributed fact bases. I.e. in the agent concept for the envisaged simulation models (fully autonomous entities, according to Gilbert and Troitzsch (2005)), each agent is planned to have its own self-contained rule engine.
- Providing an experimentation platform tailored for the design decision to implement a data-driven approach, allowing to 'play' with new functionalities, and to be able to monitor the rule engine processes. The latter point also refers to the most important requirement:

- Preserving the evidence traces from the evidence base, via the conceptual model to the simulation results.

These requirements are reflected in the architecture of DRAMS as shown in Figure 9.1.

The inference process is controlled by the Scheduler component. As introduced in section 8.3.2, DRAMS uses a data-rule dependency graph to decide which rule to consider for evaluation in a data-driven manner. The basic working principle is based upon the check which facts are available to be evaluated in the condition (LHS) part of each rule and at each event. An event is usually considered to be a 'tick' of the simulation time, where 'tick' is used here as a designator for a time unit which can be a discrete time given by a simulation timer, or a continuous point of time in an event-based approach. The evaluation is encapsulated in an action, while actions are grouped in tasks. A task contains independent actions to be processed simultaneously. Tasks themselves are then associated to an event. If the evaluation of the LHS results in a Boolean 'true', the action (RHS) part of the rule is fired.

The rule engine core of DRAMS is the managing component for the distributed rule engines. This means that each agent is equipped with private fact and rule bases. Communication between agents is either done by offering (i.e. asserting) a fact to a foreign fact base (which can only be read by the target agent), or by a global fact base, a sort of blackboard. The rule engine manager component provides these communication means. The idea behind the distributed rule engine concept is to technically support the autonomy aspect of the agent definition according to (Gilbert and Troitzsch, 2005). This approach helps to keep the single fact bases smaller and to avoid possible side effects, i.e. making the model implementation more assessable. Another function of the rule engine core is to provide interfaces to configure and use DRAMS. The agent interface allows to integrate rule engine functionality in Java-based simulation models, e.g. developed with Repast. The Result Writer interface allows to make simulation outcomes persistent or to supply the raw outcomes to other (external) components for further processing. With the UI interface, simulation results or information about the internal state of the rule engine can be monitored and visualised. All these components are implemented as plugins for DRAMS. To configure DRAMS for concrete applications the parser component allows to formulate rules, facts and all other elements in an OPS5-like language (Forgy, 1981). The architecture around the data-driven scheduler approach has a beneficial by-product: by memorising the LHS evaluations and RHS firings in a graph data structure, a full set of traces of the different steps can be produced and further processed. A demand to process such information can be two-fold: On the one hand, debugging of rule code can be facilitated; on the other hand references to evidence documents as basis of the simulation model can be preserved during the simulation process.

The following two sections are dedicated to present further details on the design and implementation of DRAMS. There a distinction is made between

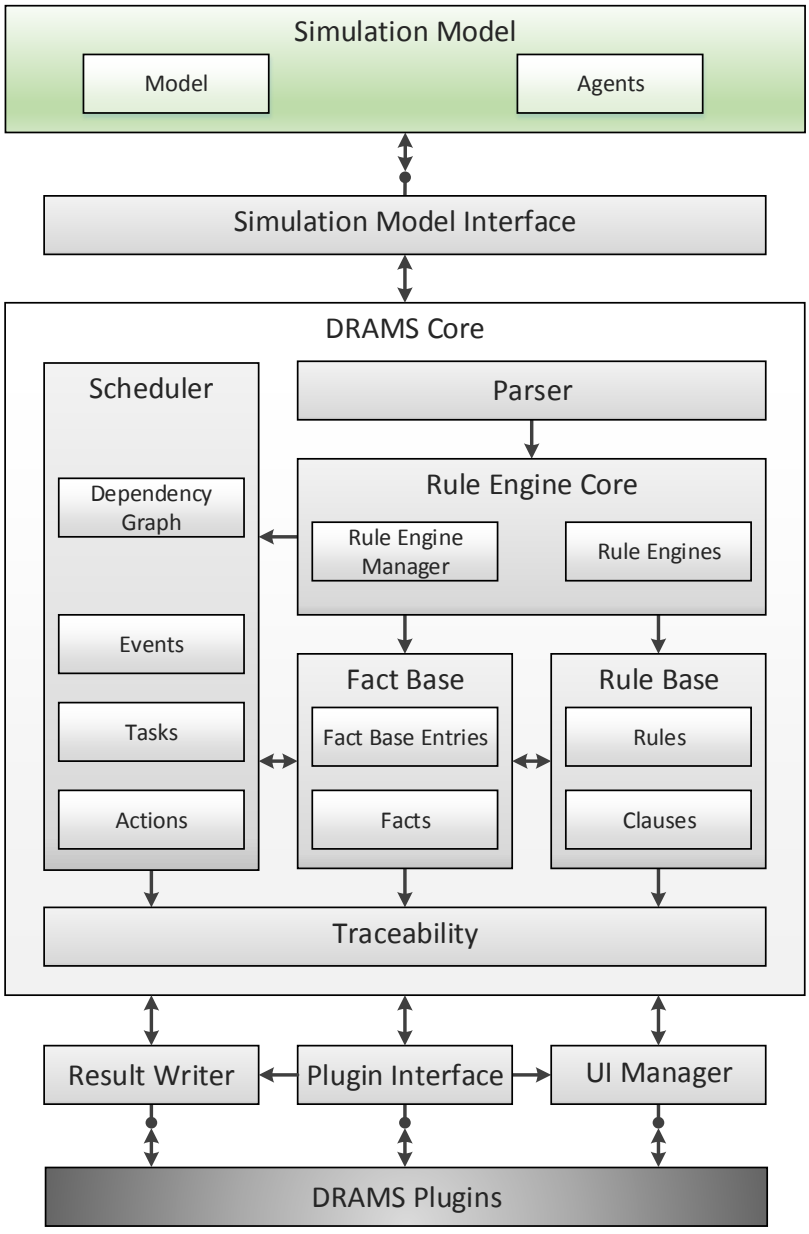

Figure 9.1: DRAMS Architecture

the core components (framed in the component diagram) and the interface components. The two external components 'Simulation Model' and 'DRAMS Plugins' are not subject of this chapter, but covered by other chapters of this PhD thesis. While an example for a simulation model is given in Chapter 6, the DRAMS Plugins are covered by the following Chapter 10.

9.3 DRAMS Core Components

Some of the components distinguished in the DRAMS core architecture of Figure 9.1 already show the most important sub-components. These sub-components refer to Java classes or class structures. The internal structure of these components and sub-components will be detailed in the following subsections. For this purpose, for each component a dedicated subsection is included, while for each sub-component a class diagram is presented. In addition, further diagrams illustrate important glue logic and processing details (by sequence diagrams). The class diagrams are equipped with some degree of implementation-related details, concentrating on the data processed by the classes, while operations are specified only for interfaces.

9.3.1 Rule Engine Core

The rule engine core contains the central control and management infrastructure of DRAMS. All other components are directly or indirectly linked with the classes involved in this component. According to Figure 9.2, there are three main types shown in an inheritance relationship: An interface `IRuleProcessingFacility`, a class `RuleEngine` implementing this interface and sub-class `RuleEngineManager`.

`IRuleProcessingFacility` defines a couple of operations for any kind of rule processing facility. There are methods to access the associated fact base and rule base, getter methods for accessing identification information (like id and owner) and the ability to be configured with declarative code (the access to a linked parser). Finally, the functionality to execute rules and declarative code is foreseen.

The standard implementation of a rule processing facility is the `RuleEngine`. This class implements the interface methods, and therefore needs to define the respective member variables, the data to be processed: the identification of the rule engine owner, the actual owner agent (identified via the `IAgent` interface, part of the simulation model interface) with meta-information like name, type and unique id; the related fact base and rule base, together with sections of the dependency graphs (containing only rules and facts managed by this rule engine).

Although the `RuleEngineManager` is a sub-class of `RuleEngine`, it is a class with a special role in the system — the central control hub and focal point for all attached system interfaces. The inheritance relation is applied because

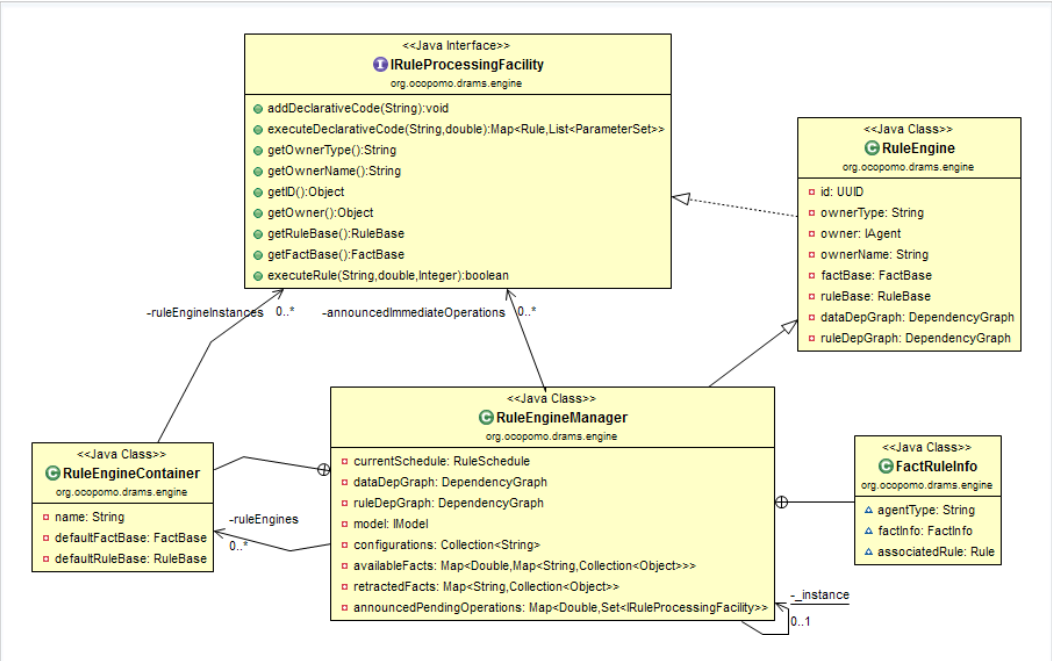

Figure 9.2: Class diagram of rule engine core

this class also stands for the representation of the simulation model (expressed by the member variable model of type `IModel`, also part of the simulation model interface). It is designed as a singleton class, i.e. exactly (formally correct 'at most') one object of this class can be instantiated. Besides the rule engine functionality, the additional duties of this class are: managing all agent-related rule engines (via the `RuleEngineContainer` helper class, where also default fact and rule bases are kept); managing the rule schedule, the heart of rule processing and the link to the scheduler component; configurations; functionality for controlling and synchronising the state of the multiple fact bases of the different agent-related rule engines (member variables for available and retracted facts and pending fact base operations and announced immediate fact base operations; and the `FactRuleInfo` helper class).

9.3.2 Fact Base

The functionality of the fact base component is to store the facts to be processed by a rule engine. It has to provide access to stored facts and must allow to modify the content. In DRAMS, for all types of facts a common interface `IFact` is defined. In Figure 9.3 this interface is shown with the methods to access the fact content:

- The name of the fact.
- Meta information, such as the owner name, a time stamp specifying the

tick at which the fact was created, and a 'permanent' flag identifying the fact to be valid at every point of time in a simulation.

- Management information, such as the fact base in which the fact is currently stored (which can be both inquired and changed), and a unique key which is used by the fact base to efficiently distinguish facts with identical content but different meta information. In addition, (human readable) declarative code for the fact can be generated.

- The actual content of the fact. This is stored in so called slots, fields with a name for which an actual value can be assigned. There is functionality to check whether a slot exists and to get the value for a particular slot.

The `IFact` interface does not foresee to change any content of the facts. This is reserved for the different implementations.

A regular fact is represented by the `Fact` class. The name and key are stored as strings, the slots in a map named `content`. Also the reference to the fact base is present, and an additional element of type `TraceTag` that keeps the traceability information (see section 9.3.6). The content of the facts is set by the constructor during instantiation of the fact. Modification of content is not possible for this fact type (with the rationale that regular facts do not change, but are replaced).

A special sub-type of `Fact` is the `ShadowFact`. Here the slots and the related content are provided by Java objects (typically the associated agent) that make the data available with getter and setter methods. These methods are kept in the two related maps. The access is effected by the Java reflection technique.

There is also another kind of shadow fact, called the `SimpleShadowFact`. With this class a fact with arbitrary name, data type `T` and other meta information can be defined and used within rules. The implementation of this shadow fact contains both a DRAMS and a Java interface, so that the value of this fact can be read and written from both declarative rules and Java code.

The actual fact base component is organised in three-layer hierarchy as shown in Figure 9.4: on the top level the class `FactBase`, whose main purpose is to store an arbitrary number of entries of type `FactBaseEntry`, whereas the fact base entry stores the actual facts, in this diagram only represented by the `IFact` interface.

The `FactBase` class provides the interface to the fact base content. It keeps information about the fact base owner (the respective agent) and hosts all related fact base entries. It provides functionality to insert fact base entries, to insert and retract facts and to perform different kinds of fact base queries. Fact base queries can be compared with the similar data base functionality, where certain data patterns can be specified, and only matching entries are retrieved. The Java signature of the standard query function is as follows:

```
public Collection<IFact> extendedQuery(
        String name,
```

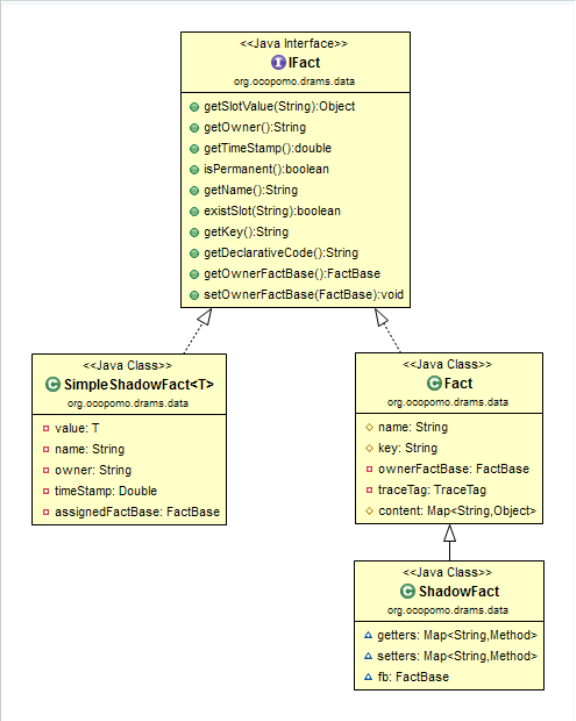

Figure 9.3: Class diagram of fact structure

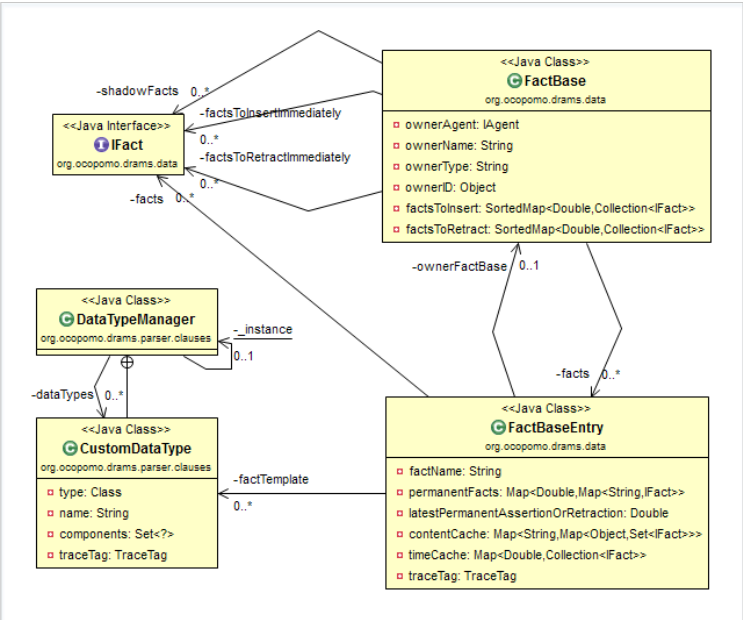

Figure 9.4: Class diagram of fact base component

```
        String owner,
        QueryTimeMode timeMode,
        Double timeStamp,
        String[] slots,
        Object[] values
)
```

As for all queries, the return type is a collection of `IFact`-compatible objects. This collection contains all facts that match the pattern specified by the parameters of the signature:

- The names of fact and owner (i.e. agent name);
- a temporal condition consisting of a `timeStamp` value and an associated `timeMode`. With the latter interpretation of the specified time stamp can be influenced, e.g. whether the facts have to be inserted at exactly the time stamp, or before, or after. Some time modes do not require a time stamp value, e.g. for the last or the latest asserted facts. All available time modes are specified in the enumeration `QueryTimeMode` and documented in (Lotzmann and Meyer, 2013).
- Arrays of slot names and associated values. Only facts are retrieved whose content for the specified slots equal the specified values.

Any parameters assigned with a `null` value are not regarded in the query.

Another more comprehensive query allows different compare operators for the slot matching pattern. The Java signature of this method is as follows:

```
Collection<IFact> cachedExtendedQuery(
        String name,
        String owner,
        QueryTimeMode timeMode,
        Double timeStamp,
        Map<String, AbstractSlotComparator> slotComparators
)
```

Return type and meta data is the same as for `extendedQuery`, but this time the slot pattern is represented by a map where for each slot name an `AbstractSlotComparator`-compatible object is specified. This class is shown in Figure 9.5 together with actual implementations and helper classes (based on the `ISlotValueProvider` interface) for providing the values.

There are two implementations for the slot value provider: The `ConfigurationSlotValueProvider` is used to compare the current slot with a constant value stored in this provider (similar to the values array in the `extendedQuery`). The `FactSlotValueProvider` on the other hand can access another slot value that is compared with current slot as specified by the map key.

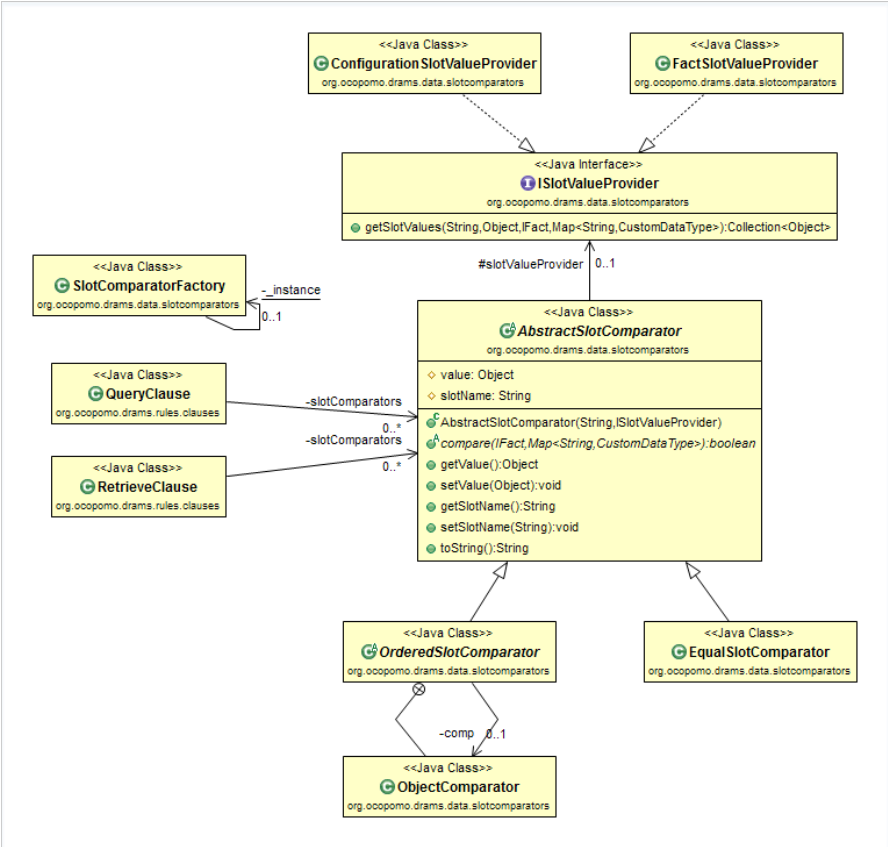

Figure 9.5: Class diagram of slot provider helper classes

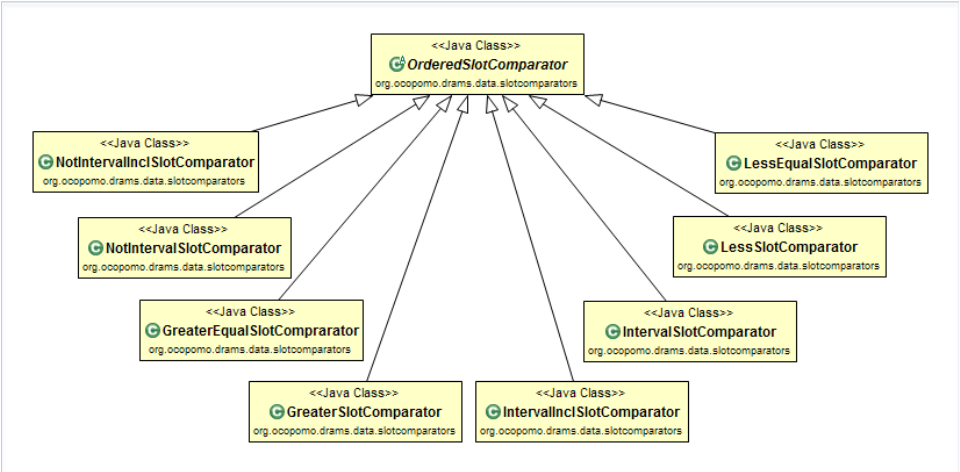

Figure 9.6: Class diagram of concrete ordered slot comparator classes

The `AbstractSlotComparator` uses the data supplied by the slot value providers when evaluating rules, concretely `QueryClause` or `RetrieveClause` objects (these will be detailed in the following section 9.3.3). Several concrete implementations of `AbstractSlotComparator` are generated by a `SlotComparatorFactory`: One implementation for checking for equal content (the `EqualSlotComparator`; i.e. the identical functionality as provided by `extendedQuery`), and a couple of implementations for `OrderedSlotComparator` (utilising an `ObjectComparator` helper class) according to Figure 9.6:

- comparative operators $<, \leq, >, \geq$;
- set operators $\in$ and $\notin$.

9.3.3 Rule Base

The rule base can be considered as a counterpart to the fact base, where the rules are stored to be processed by the rule engine. Thereby the rules read from and write to the fact base, i.e. the fact base is the working memory for the rules. The class diagram in Figure 9.7 gives an overview on the rule base component.

The `RuleBase` class itself has a quite simple structure: It keeps memory of the associated owner agent and a collection of rules, represented by the `Rule` class.

The construction of the `Rule` class is mainly oriented on the structure of a rule, consisting of a component representing the left-hand side (LHS) and another component for the right-hand side. Besides the associations to the related two classes `LHSComponent` and `RHSComponent`, other member variables hold information for different purposes. So the rule can be identified by a name (a unique designator), and a link to the rule base to which the rule belongs

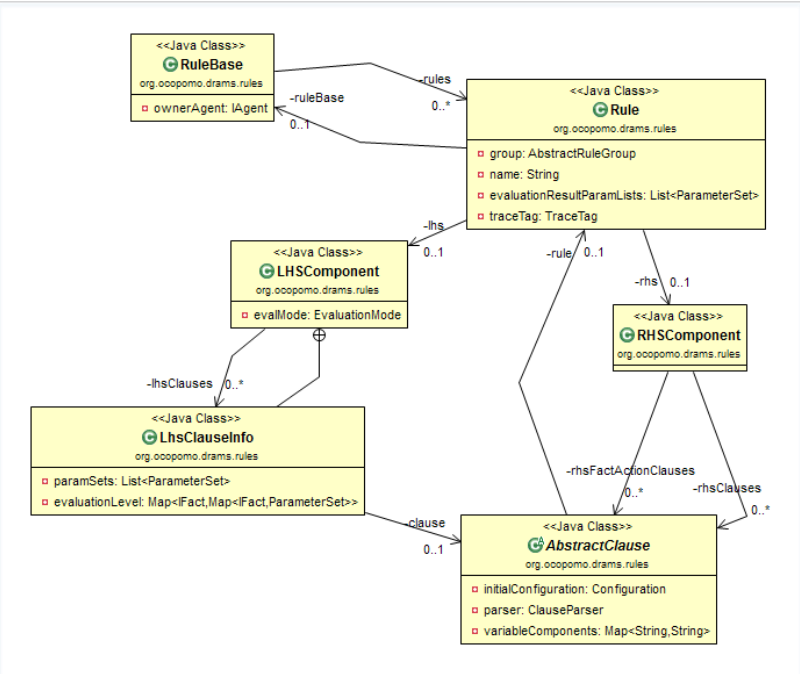

Figure 9.7: Class diagram of rule base component

is also present. Further on there are member variables holding information to control the evaluation process: A facility to organise rules in so-called rule groups (represented by the class `AbstractRuleGroup`) and a list for `ParameterSet` to store the evaluation results of the LHS, i.e. the values bound to variables for all cases of successful LHS evaluation. The functioning of the evaluation procedure and the interplay of the different classes are described in the following subsection Scheduler. Finally, traceability information is stored by a `TraceTag`.

The `AbstractRuleGroup` allows to define different kinds of dependencies between the rules subsumed under the rule group. Figure 9.8 shows the related class structure. Apart from the means to add rules to the group, it offers functionality for rule engines (expressed by the `IRuleProcessingFacility` interface) to check whether a successful evaluation of the incorporated LHS's is possible at a certain tick and, most importantly, controls the execution of the rules in the group. By default (i.e. if no group type is specified) the rules are in an OR-relation, meaning that any arbitrary number of rules can fire. Here the concrete implementation `RuleGroupOR` is used. There are two alternative ways to define relations between rules: either in an AND or in an XOR relation. Rules in an AND group (class `RuleGroupAND`) fire all together at the same time, (only) if the LHS's of all rules have been successfully evaluated. In contrast, at most one rule fires in an XOR group (class `RuleGroupXOR`), no matter how many LHS's have been successfully evaluated.

The structure of the two rule components for the LHS and RHS is quite

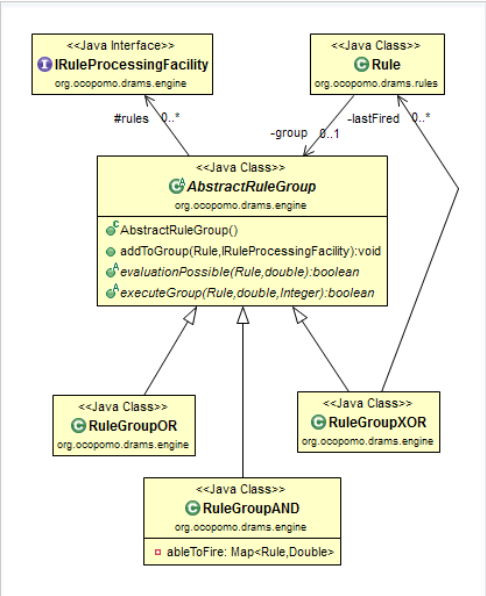

Figure 9.8: Class diagram of the rule group classes

different due to the unequal complexity of these two rule parts. While for the RHS it is sufficient to keep a collection of RHS clauses (class `AbstractClause`; here for technical reasons split into a collection for clauses regarding fact base operations and a collection for other clauses; see Figure 9.10) to be executed in case the rule fires (i.e. he LHS evaluation is successful), the LHS component is more complicated. This becomes visible by a member variable specifying an evaluation mode and by the intermediate helper class `LHSClauseInfo` that wraps each LHS clause in order to keep data for LHS evaluation ready, a list of parameter sets and information about evaluation levels (again, this will be detailed in subsection Scheduler).

The basic principle of LHS evaluation is as follows: All clauses of the LHS are evaluated in the order of appearance. The evaluation of each clause returns a boolean value. Depending on the evaluation mode (which has nothing to do with the rule group introduced above as here only the LHS is affected), the LHS evaluates successfully, if all (AND mode), at least one (OR mode) or exactly one (XOR mode) of the LHS clauses return true, stored in an object of type `EvaluationResult`. In addition, the `EvaluationResult` class also maintains a set of variables defined for the rule (encapsulated in the class `ParameterSet`) is passed through the different clauses. Each clause can access the parameter (constants or variables) defined in the set, and write new values to unbound variables, i.e. parameters that are not already assigned with a value. If a clause results in more than one valid variable assignments, then all these different parameter sets are passed over to the next clause. Hence, the evaluation of all clauses generates an evaluation tree, with its leaves as the final collection of parameter sets as the evaluation result. This collection is then the base for firing the RHS.

The actual operations collected by a rule are represented by clauses, defined by the `AbstractClause` class. This class keeps a relation to the rule of which it is part of, and to further data: information for processing the clause (an initial configuration for variables included in the clause with additional information on components in case of composite variables, e.g. to access a certain slot of a fact) and a link to an associated parser. The are a number of concrete clause implementations, some dedicated for either the LHS or the RHS, and others valid for both parts of the rule. The assignment to the LHS or RHS is done with the marker interfaces `ILHSClause` and `IRHSClause`, which the clauses implement (in addition to extending the AbstractClause[106]).

All clause classes implementing the `ILHSClause` interface are shown in Figure 9.9 and briefly outlined in the following paragraphs.

Retrieve clause The `RetrieveClause` class is the standard clause for retrieving facts from the fact base. It encapsulates the fact query functionality of fact base as described in subsection 9.3.2. The identification of the fact to access is done by the `factDescriptor` member variable. The search pattern for the fact base query is stored in the `slotComparators` map data structure with the predefined slot comparators for the individual slots, temporal conditions are set by `lagMode`, `lagValue` and `lagRelativeValue`. Unbound variables specified in the slot comparators are bound with values from the retrieved facts. If more than one fact fulfils the specified pattern, then for each of the facts potentially a new parameter set is generated, containing different variable assignments.

`RetrieveClause` has a sub-class `ExistsRetrieveClause` that does not assign any variables from the retrieved facts, but just yields a boolean result. Hence, for both classes the result of the retrieve clause is true, if at least one fact matches the specified pattern.

QueryClause The `QueryClause` class covers the same functionality as the retrieve clause, i.e. it encapsulates the fact base query functionality. In contrast to the fact base retrieve clause, the query clause is typically used with a result variable, i.e. a variable that is bound with the query result. The query result is a collection of the facts matching the specified pattern, and in case of a successful evaluation, just one parameter set is generated, independent from the number of matched facts.

Java action clause The `JavaActionClause` class encapsulates several general-purpose functions with relation to the Java-part of the simulation model, specified in the `Statements` enumeration:

- `PRINT` writes a text to the simulation log. This is per default printed to the Java output stream, but can be relocated to specific Result Writers

[106]This is the only way of multiple inheritance foreseen in Java.

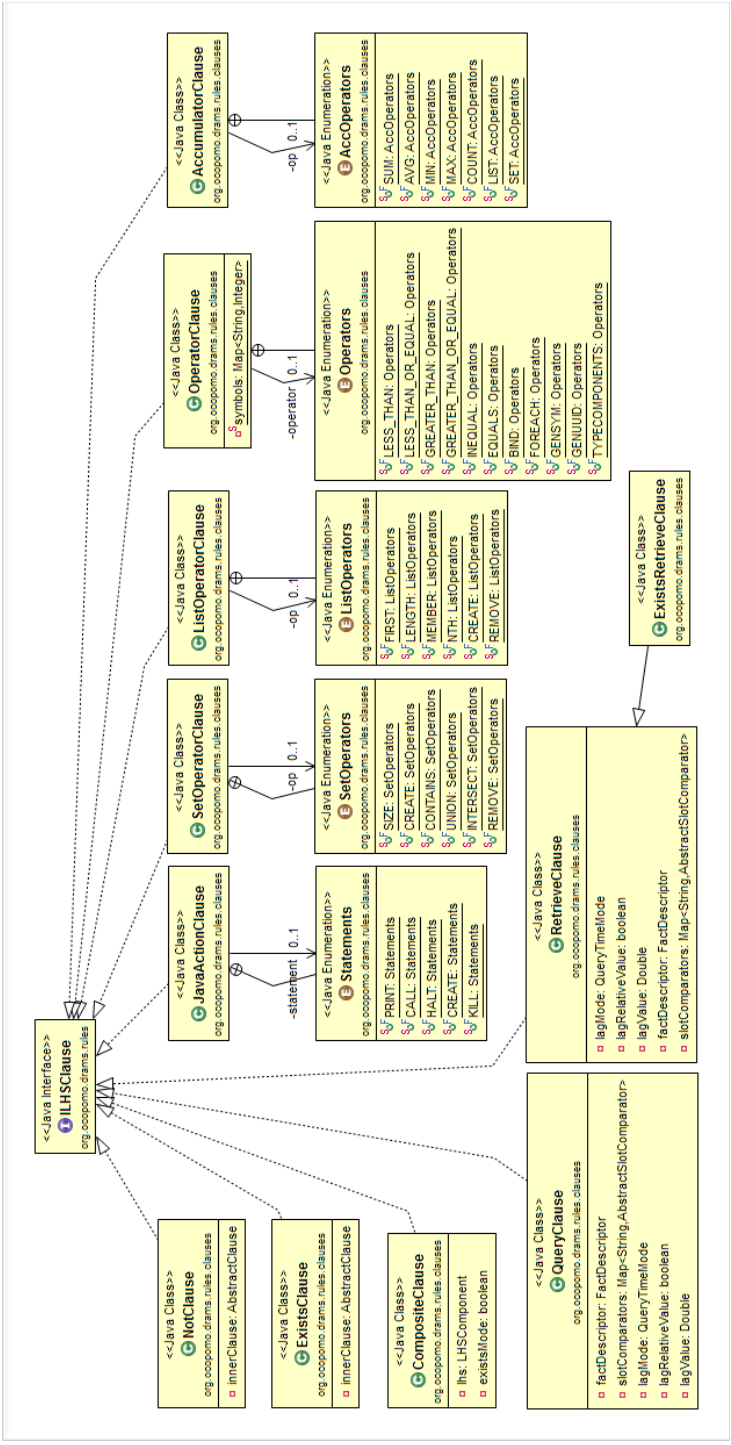

Figure 9.9: Class diagram of LHS clauses (remark: super class `AbstractClause` for all clause classes not shown in diagram)

(as defined in section 9.4: Result Writer). The string can contain variable names that will be replaced by actual variable assignments at printing time. Also a priority parameter is supported to direct outputs of different priorities to different Result Writers. The evaluation result is always true.

- `CALL` invokes a Java method of the associated model agent class. The method signature can be specified for the class, the Java parameter can be mapped from DRAMS variables, and DRAMS variables specified as part of the evaluation result can be bound by the Java method result.
- `HALT` can be used (mainly for model debugging purposes) to pause the timer of the applied Java simulation tool (e.g. Repast). Result is always true.
- `CREATE` triggers the 'birth' of a new agent instance of the specified type with the specified name.
- `KILL` destroys the specified agent instance.

Set-operator clause The `SetOperatorClause` class encapsulates several operators to be applied on or in relation to sets, specified in the `SetOperators` enumeration:

- `SIZE` returns the number of elements in a set.
- `CREATE` constitutes a new set from a given set or list (can be also specified as a set constant or an expression).
- `UNION` combines two sets.
- `INTERSECT` generates the intersection of two sets.
- `REMOVE` deletes an element or elements from another list/set from a set.

List-operator clause The `ListOperatorClause` class encapsulates several operators to be applied on or in relation to lists, specified in the `ListOperators` enumeration:

- `FIRST` returns the first element of a list.
- `LENGTH` returns the number of elements in the list.
- `MEMBER` tests whether a specified element is part of a list.
- `NTH` returns the n-th element of a list.
- `CREATE` constitutes a new list from a given list or set (can be also specified as a set constant or an expression) or extends an existing list by new elements.
- `REMOVE` deletes an element or elements from another list/set from a list.

Operator Clause The `OperatorClause` class encapsulates a number of general-purpose operators (both unary and binary), specified in the `Operators` enumeration:

- Several binary compare operators with boolean result: `LESS_THAN` for $<$, `LESS_THAN_OR_EQUAL` for $\leq$, `GREATER_THAN` for $>$, `GREATER_THAN_OR_EQUAL` for $\geq$, `INEQUAL` for $\neq$ and `EQUALS` for $=$.
- `BIND` binds an unbound result variable with the specified value or expression.
- `FOREACH` creates several bindings of the result variable with the elements of the specified list.
- `GENSYM` binds an unbound result variable with a symbol. A symbol contains of a specified string with an attached unique sequential number.
- `GENUUID` binds an unbound result variable with an unique identifier (UUID).
- `TYPECOMPONENTS` creates several bindings of the result variable with the components of the specified enumeration data type.

Accumulator clause The `ListOperatorClause` class encapsulates several accumulative operators to be applied on lists of facts (as generated by the query clause), specified in the `AccOperators` enumeration:

- `SUM` calculates the sum of the slot values.
- `AVG` calculates the average value of the slot values.
- `MIN` returns the minimal slot value.
- `MAX` returns the maximal slot value.
- `COUNT` returns the number of different slot values existing in the fact list.
- `LIST` generates a list with the slot values for all facts is the fact list.
- `SET` generates a set with the different slot values existing in the fact list.

For all operators the fact list and the slot name over which the accumulation should be performed has to be specified.

Not clause The `NotClause` class is a wrapper for other clauses that simply negates the evaluation result for the inner clause.

Exists clause The `ExistsClause` class is a wrapper for other clauses that passes through the boolean evaluation result but discards all variable bindings done by the inner clause.

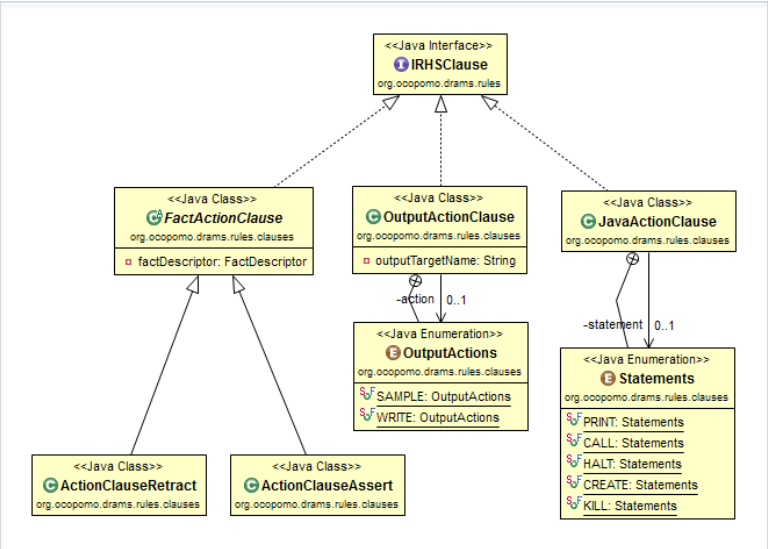

Figure 9.10: Class diagram of RHS clauses (remark: super class `AbstractClause` for all clause classes not shown in diagram)

Composite clause The `CompositeClause` class is a wrapper for a sub-LHS, i.e. a number of LHS clauses can be grouped in such a composite. As for each LHS, the evaluation mode (AND, OR, XOR) can be specified, so that an evaluation logic can be defined for systems of LHS clauses. The `existsMode` member variable refers to the availability of variable binding outside the inner LHS.

The RHS clauses implementing the `IRHSClause` are summarised in Figure 9.10. The following paragraphs give an overview on these classes. They all carry the word 'action' in the name, referring to the nature of the RHS.

Fact action clause The abstract `FactActionClause` class is the template for classes implementing writing fact base access. They can be seen as the counterpart to the retrieve and query clause classes for the LHS; here, also the fact descriptor exists as a member variable, but other parameters (like e.g. the content of the facts to write or temporal conditions for deferred fact insertion) are not stored in the class as such information is contained in the LHS result parameter set handed over to the RHS. Two different implementations are included, on the one hand the `ActionClauseRetract` for deleting existing facts, on the other hand `ActionClauseAssert` for creating new facts.

Output action clause The `OutputActionClause` encapsulates functions for advanced (over text logging) generation of simulation outcomes. It maintains a link to a Result Writer (the `outputTargetName`) and has means to

create tabular data sets with accumulated data. As defined by the `OutputActions` enumeration, there are two functions: `SAMPLE` collects data which is then written to the Result Writer with the `WRITE` function. For example, a row of the table is first *sampled* and — when complete — *written* to e.g. a CSV table.

Java action clause This is the same class as the Java action clause described for the LHS above, see Java action clause. Hence, this is the only clause class that implements both the `ILHSClause` and `IRHSClause` interfaces.

9.3.4 Scheduler

The scheduler is the core component for the rule engine dynamics and, thus, by far the most complicated part of DRAMS. The requirements for this component not only refer to guaranteed logically correct processing of rules and fact base management, but also set constraints for reasonable runtime performance and preservation of traceability information. The functionality of this component is spread over many classes — some of them have been mentioned before in the context of fact or rule base — with manifold relations in-between. These relations will be illustrated in context of the interaction dynamics with the help of a sequence diagram. But before the static aspects shall be addressed.

Main class of the scheduler is `RuleSchedule`. Figure 9.11 shows this class together with classes representing the events to schedule. As described in section 8.3.2, the function of the scheduler is to select the rules to be evaluated

- for each tick (i.e. point of simulation `time`, represented by the respective member variable, together with a shadow fact `timeShadowFact` that enables access to the current simulation time from within rules), and
- for each configuration of the fact bases (i.e. state of the simulation model).

Thereby it has to cope with several constraints (mainly for optimisation of runtime performance), such as to pre-select all rules for which a chance for successful evaluation of the LHS exists (the `possiblyFiringRules`), and at the same time to consider dependencies between these rules. So, for example, if two rules A and B could be fired at the same time, but rule A (in case it fires) asserts facts that might constitute new (additional) preconditions for rule B, then the scheduler takes care that first rule A is evaluated and subsequently rule B. The overall aim for a scheduler for a data-driven approach is to find an optimal order of rule evaluation. For this purpose the scheduler draws on the data dependency graph, represented by the `DependencyGraph` class, described below. Some details of the technical realisation (hence, slightly off-topic for this chapter) of the scheduler are the `initialNotClauseScheduled` member variable to cope with clauses that are wrapped in a not-clause, and the inner

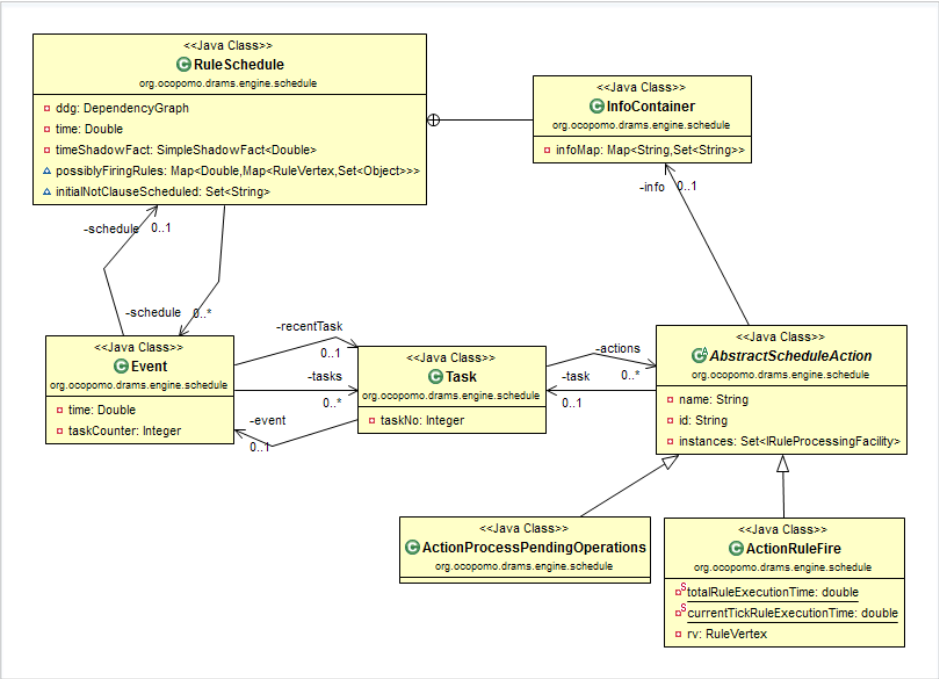

Figure 9.11: Class diagram of scheduler component

class `InfoContainer` used to compile scheduler status logs (displayed in the user inter face as shown in Figure 7.8 of section 7.3.2).

Here a peculiarity of DRAMS again comes to light, the nested time structure as illustrated in Figure 8.2 of section 8.3.2. In the class diagram in Figure 9.11 this is reflected by the differentiation between the `Event` class that stands for a tick (i.e. a point in simulation `time`, the respective member variable) and the `Task` class that encapsulates a (numbered) 'moment' within a tick. The `Event` keeps a collection of all associated tasks as well as a reference to the task most recently added to the collection. The Task just collects the actions to be ultimately performed by the scheduler.

The actions are represented by the `AbstractScheduleAction` class. Each action has a name and an identifier, and its main functionality is to host a set of rule engines (expressed by the `IRuleProcessingFacility` interface) to which the action is related. The abstract class is implemented by two concrete classes. The `ActionProcessPendingOperations` class triggers the processing of pending fact base operations, e.g. writing facts to fact bases that were asserted during the previous task. The actual rule evaluations are scheduled by the `ActionRuleFire` class. Besides (static) variables to measure the performance of the rule processing (rule execution time related), this class references to a `RuleVertex`, a node of the dependency graph that represents the rule to process.

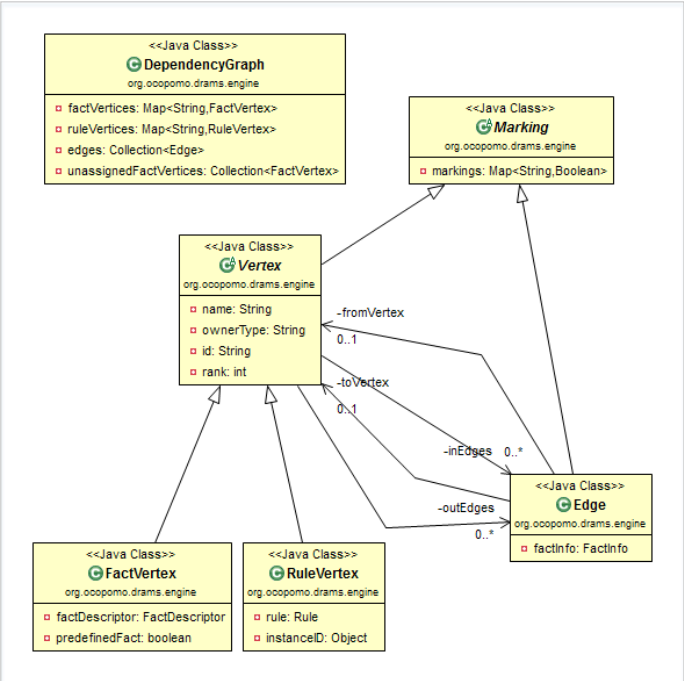

Figure 9.12: Class diagram of dependency graph

Dependency Graph

The dependency graph class structure is shown in Figure 9.12. The graph realisation follows a simple straight-forward object-oriented approach. The abstract class `Marking` is the super class for `Vertex` and `Edge` classes. The map data structure `markings` in `Marking` is used to flag a marking with boolean attributes that are needed for the applied graph operations (e.g. for finding cycles or extracting sub-graphs).

The (abstract) `Vertex` class represents the nodes of the graph. Each vertex stores name, identifier, type of owner (the agent type for which the related rule is valid) and a rank value. The rank (which is not related to any rank notion of rank in graph theory) has a particular importance for another graph operation related to the scheduler: the calculation of a rule dependency graph, a directed acyclic graph showing the hierarchy of rules from a given starting point (a rule vertex).

Two different kinds of vertices are distinguished, represented by the concrete classes `FactVertex` and `RuleVertex`. The former provides the link to an actual fact (template) via a `FactDescriptor`, a class assembling complete access information for facts stored in fact bases. The latter holds a link to an actual rule.

`Vertex` stores its adjacent nodes as in- and outgoing edges. An edge — represented by the `Edge` class — is a directed relation, i.e. the linked vertices are referred to as `fromVertex` and `toVertex`. This class also contains a `factInfo` member variable that enables access to information regarding the

dependency between fact and a rule linked by the edge, like e.g. the fact descriptor or timing information (lag mode and value for fact retrievals, deferred fact assertion).

Given the way adjacent markings are represented, the dependency graph can be characterised as directed graph that allows algorithms to traverse the graph bi-directionally. The class `DependencyGraph` is the entry point to the graph data structure by defining the basic algorithms and access functionality. Therefore it not only encapsulates data structure for (fact and rule) vertices and edges, but also a collection of unassigned fact vertices (facts, for which the owner only becomes known during runtime) for optimisation of algorithms.

An illustration of a dependency graph is shown in Figure 7.11 in section 7.3.4. From this illustration and the description above, the dependency graph used here can more concretely be termed as a data-rule dependency graph, since facts and rules are alternately concatenated. This structure is closely related to the CCD action diagram, the approach to conceptually modelling dynamic aspects of a system to simulate proposed by OCOPOMO (see section 3.3.2). Once the dependency graph is created it is used to control the process of rule evaluation and processing. These two core processes of DRAMS are detailed in the following two paragraphs, both from the static and the dynamic viewpoint.

Schedule initialisation

The initialisation of the schedule basically comprises the transformation of the rules, fact templates and facts (the entities defined by the declarative code) into a dependency graph. The sequence of interactions between different classes involved in the initialisation process is sketched in Figure 9.13. The initialisation is triggered by the simulation model, here represented by the `Model` class. An abstract model main class is provided by DRAMS (see subsection 9.4.1), that defines a `begin()` method to create the schedule. As the name of the method suggests, schedule initialisation is typically done as a first procedure after starting a simulation run. But also under other circumstances and at later points of time an initialisation is performed, e.g. for the on-the-fly rule processing from the DRAMS console, see section 7.3.3.

The `begin()` method resorts to the `getSchedule()` method of the rule engine manager. The first step is the set-up of the dependency graph (method `createDataDependencyGraph()`) from the rules stored in rule bases. For each rule, a rule vertex is created, all input facts on which a rule depends (i.e. that are retrieved by the rule's LHS) are linked with the rule, meaning that edges are inserted between the rule vertex and the fact vertices representing the input facts. If for a fact no vertex is present, a new fact vertex is created. The similar process is repeated for the output facts, i.e. the facts that are asserted by the rule's RHS.

From the full dependency graph the schedule is then derived. The

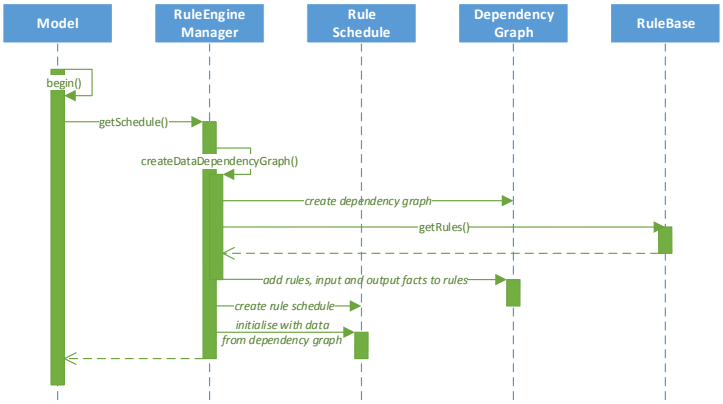

Figure 9.13: Sequence diagram of schedule initialisation

`RuleSchedule` object is created and initialised by basically creating a first set of possibly firing rules. The primary schedule event has to be treated differently from all other (following) events in a data-driven approach where fact base operations determine the rules to schedule, since at the initial fact base state no operations are present. So, for example, even with completely 'empty' fact bases rules might fire that evaluate non-existing facts (not-retrieve or not-exists). Furthermore, the distributed rule engine approach of DRAMS demands a selection of all the rule engine instances that contain possibly firing rules. In typical agent-based systems, there are many instances of the same agent type, which all have independent rule engines assigned. The agent-individual fact base states, on the other hand, not necessarily require all engines to schedule a rule that might fire for just a single instance (where the pre-condition facts are available). This complicates the processing, also for the rule processing following the schedule initialisation.

Rule processing

The rule schedule as the basis for rule processing for each event needs to be updated on different occasions in order to evaluate the right rules at the right point of time. For the round-based time mode the schedule is updated at each tick, while for the discrete event mode this procedure is done only at (previously) scheduled events.

For the former case, the processing of the rules for a tick follows the scheme as shown in Figure 9.14. The difference to the discrete event mode is marginal, since for the latter the event with the lowest timestamp is taken from the schedule.

For processing a tick, the model class defines an `execute(tick)` method, which invokes the `processNextTick()` method of `RuleSchedule` with the current tick as parameter. `processNextTick()` performs the following consecu-

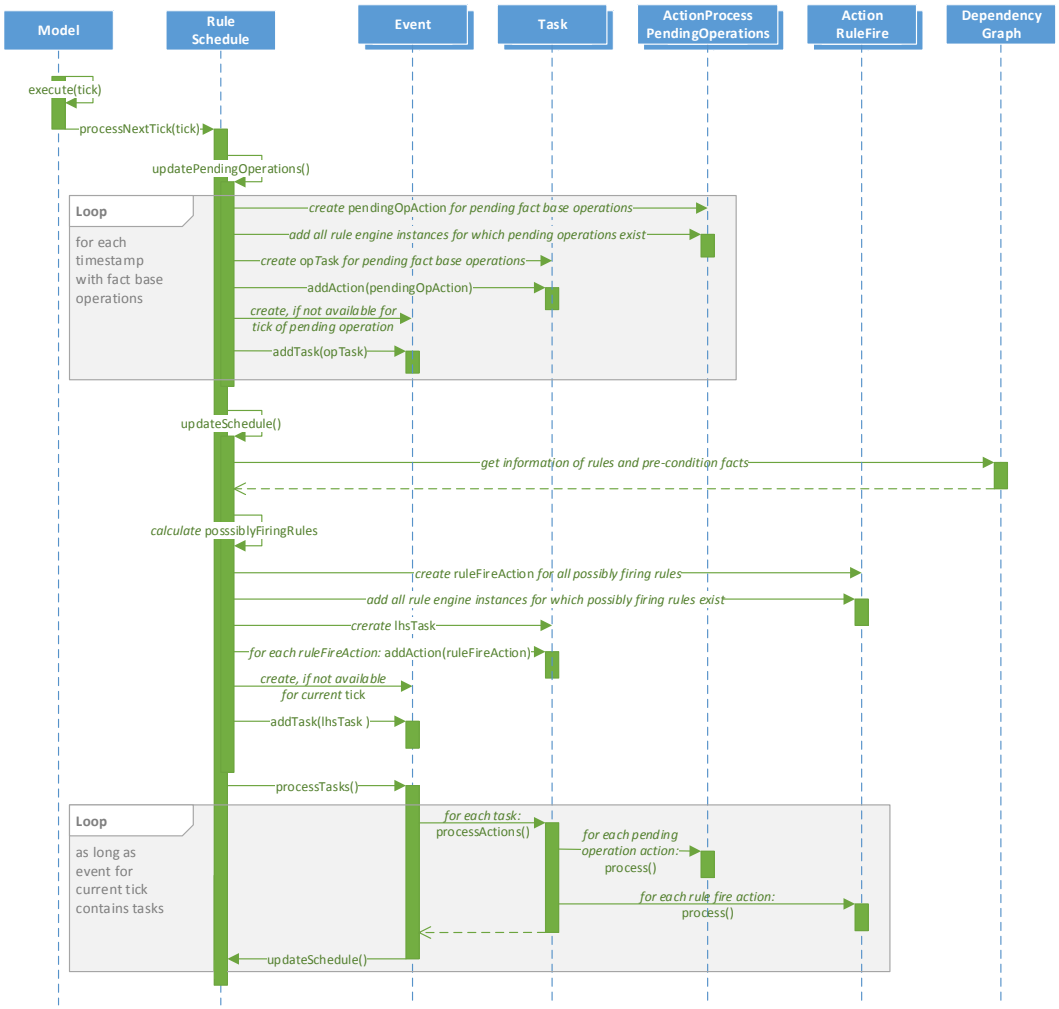

Figure 9.14: Sequence diagram of tick execution

tive steps:

1. Pending fact base operations are added to the schedule (method call `updatePendingOperations()`). For example, a rule processed at tick t may assert a fact with a delay, say at tick $t + x$. This causes an action to be scheduled at task 0 for tick $t + x$.

   This is realised by a number of steps that are repeated for each timestamp of the fact operations, i.e. the tick at which a fact base operation should be executed. Firstly, a new object for `Action-ProcessPendingOperation` is instantiated and configured with all rule engine instances for which pending fact base operations are registered. This action is then added to a newly instantiated `Task` object. This task is then added to the event of the tick that matches the timestamp of the fact operation. If no `Event` object exists for this tick yet, a new instance is created.

2. The schedule is updated with possibly firing rules at the current tick for the next task (method call `updateSchedule()`).

   As first step all rules and pre-condition facts are retrieved from the dependency graph and inspected for possibly firing rules. An optimised algorithm identifies the rules that should be added to the schedule by matching the LHS clauses of rules with the existence of respective pre-condition facts. This step is done without taking the actual content of the facts into consideration, i.e. it serves as a fast pre-selection of all rules for which the necessary conditions for a firing are constituted.

   In the next step all rule engine instances for each possibly firing rules are identified and added to a newly created instance of `ActionRuleFire`. These action objects are then added to the task to be processed next. If no respective `Task` object exists, a new instance (called `lhsTask` in the diagram) is created.

   Finally, the `lhsTask` can be added to the event of the current tick.

3. The pending tasks for the current tick are processed. This involves the tasks for fact base operations and for possibly firing rules created just before, but also tasks of the former kind scheduled at previous tasks.

   The processing is triggered by the rule scheduler by invoking the method `processTasks()` method of `Event`. There, for each task the method `processActions()` of `Task` is invoked. The actions assigned to a task are finally triggered by invoking the `process()` method of all `Action-ProcessPendingOperations` and `ActionRuleFire` objects. Hence, the processing is done straightforward, i.e. by exploiting the intrinsic functionality of the involved classes in a nested manner.

The `ActionProcessPendingOperations` performs physical fact base operations, i.e. writes or deletes facts from the memory. The `ActionRuleFire`, on the other hand, is responsible for the entire rule processing: Firstly, the evaluation of the LHS, and secondly the execution of the RHS of the rule (i.e. the firing of the rule).

4. After all current tasks are processed, the `Event` calls the `updateSchedule()` method of `RuleScheduler` (i.e. the procedure described in step 2), herewith ensures that the schedule is updated on base of the results from the rules just fired. This step is done repeatedly until no more new tasks are pending for the current event. This also concludes the processing of the respective event.

In step 2 above a fast pre-selection of possibly firing rules is made. These rules have then to be further evaluated and possibly fired, which happens as part of step 3. The evaluation of the LHS is a central matter of DRAMS, hence this process is further detailed in the following. A number of classes is involved in this context; Figure 9.15 gives an overview.

The LHS is represented by the class `LHSComponent`, associated to the `Rule` class. In the diagram there is also the enumeration `EvaluationMode`. Here the three modes AND, OR and XOR are defined in which the clauses of the LHS can be logically linked, a setting controlling the LHS evaluation process. The LHS clauses (here represented by the abstract super class `AbstractClause`) are linked to the LHS via a `LhsClauseInfo` class, a wrapper that basically supplies the data to be processed by the LHS. The data consists of variables that can — in the context of the LHS evaluation — be bound (i.e. filled with content) via fact base retrievals, manipulated by various operations, used again as source for new fact base retrials as part of search pattern, and — in the context of the RHS — are finally constituting the basis for firing the RHS. The variables are contained in the `ParameterSet` class as a collaborator to the `LhsClauseInfo`. The `ParameterSet` itself sources information to the `ParameterInfo` class which is used for monitoring the evaluation process and collecting respective meta information, e.g. for profiling and visualising the evaluation tree (an example that processes this information is introduced in the following Chapter 10). The result of each clause evaluation is then laid down in the `EvaluationResult` class. In addition, the `Configuration` class plays an important role in the processing of the actual clauses. On the one hand it is equipped with initial information from the rule definition, such as constant values and (mathematical) expressions, on the other hand it performs the actual calculation of concrete operations defined by the clause, e.g. evaluating an expression with actual variable configuration. It relies on the functionality provided by the `ExpressionInfo` class, a wrapper for the mathematical expressions defined here by the `AbstractMathExpression` class.

The interplay of the classes of Figure 9.15 is shown in the sequence diagram in Figure 9.16. The LHS evaluation is triggered by the rule group (i.e. one

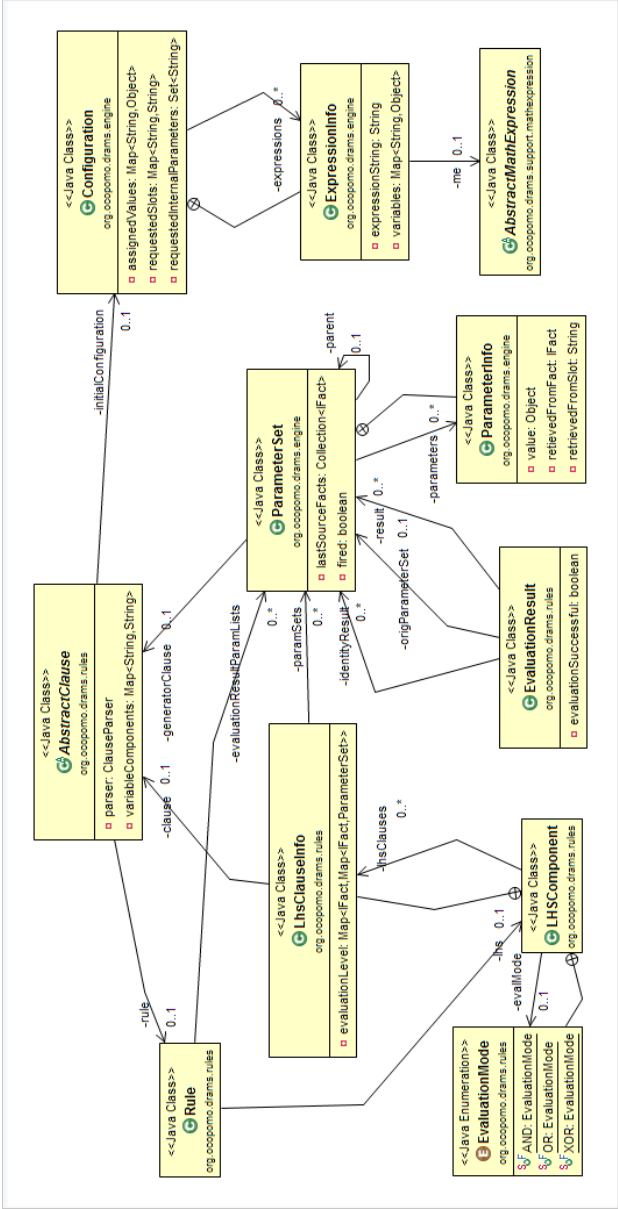

Figure 9.15: Diagram of classes involved in LHS evaluation

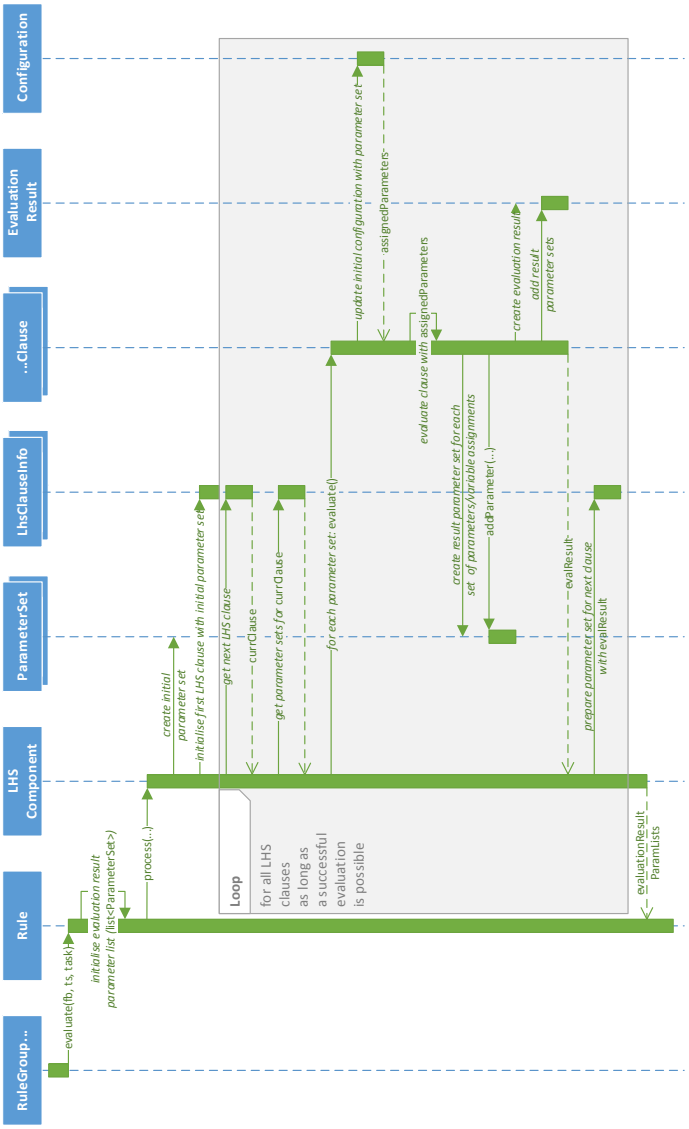

Figure 9.16: Sequence diagram of LHS evaluation

of the classes implementing the `AbstractRuleGroup`) by invoking the `evaluate()` method with parameters defining the fact base to be used, and the time stamp and task for which the evaluation should take place. The first action of this method is to initialise the evaluation result parameter list, a collection of `ParameterSet` objects that will hold the final results of the LHS evaluation. Subsequently, the processing is taken over by the `LHSComponent` class's `process()` method. This method firstly creates an initially empty parameter set and hands it over to the first LHS clause in the sequence, expressed by an `LhsClauseInfo` object.

The following procedure is then repeated for all LHS clauses as long as a successful evaluation can be expected, i.e. (depending on the evaluation mode) a final Boolean result 'true' is possible. In AND mode for example, if any clause returns a 'false', then the evaluation of this branch stops and returns a final result 'false'. The procedure consists of several steps:

1. The next untreated LHS clause (`LhsClauseInfo` object) is fetched.

2. The parameter sets filed for this clause are fetched. In case of the first clause, just the initial parameter set (as initialised before) is available, in case of all other (subsequent) clauses the parameter sets are generated by the previous clause evaluations. If a clause becomes true in different situations, for example multiple facts match the search pattern in a fact base retrieval, for each of these situations an individual parameter set is made available to the following clause.

3. For each parameter set the method `evaluate()` of the actual clause object is invoked (here labelled with `...Clause`).

   (a) The information of the current parameter set is fed into the initial configuration, i.e. possibly existing variable placeholders contained in the configuration are substituted with actual content from the parameter set. Mathematical expressions included in the configuration are calculated, and the resulting assigned parameters are returned.

   (b) The clause is evaluated with the assigned parameters.

   (c) For each evaluation with a Boolean result true, any newly assigned (bound) variable is stored (via `addParameter()` method) in a newly created `ParameterSet` object, as described above.

   (d) An `EvaluationResult` object is created and filed with the parameter sets created just before.

   (e) The Boolean result is returned to the `LHSComponent`.

4. The resulting parameter sets are prepared for the next clause, i.e. handed over to the respective `LhsClauseInfo` object.

5. The evaluation result handed back to the `LHSComponent` is then used to prepare the parameter sets for the following clauses.

Finally, the overall LHS evaluation result is returned to the `Rule` and the `RuleGroup`, from which then the RHS can be fired with all valid `EvaluationResult` objects.

9.3.5 Parser

Up to this point the functional components of DRAMS have been introduced: The rule engine and its evaluation mechanisms as static parts of DRAMS, and the fact and rule bases which can be configured for the desired purpose and application. The configuration can be achieved by instantiating the objects for fact template, fact, rule and clause classes with the wanted content and properties.

In order to increase the usability of DRAMS, a parser component provides means for the simulation modeller to *program* the rule engine configuration. As programming language the OPS5 approach was adapted, as discussed before. The translation of OPS5 programming language constructs into DRAMS objects is done by a hierarchy of dedicated parsers as depicted in Figure 9.17, based on the JavaCC[107] framework. The basic working principle of JavaCC is the automatic generation of Java code of a LL(k)[108] parser from a grammar. The grammar contains the syntax definition and Java code fragments for processing parsed language constructs.

The idea behind this kind of distributed parser has a two-faceted background:

- On the one hand to allow 'mixed' approaches to configure the system. For example, rules might be implanted from another system by creating `Rule` objects; the clauses constituting the 'innards' of the rule (i.e. the logic) are presented by OPS5 code. A (perspective) application scenario for this example might be a graphical design tool where the rule structure can be interactively 'clicked together' while the condition logic in attached as OPS/5 statements.

- On the other hand to keep the grammar simple so it can be easily and independently be replaced and extended for the various configurable elements of DRAMS.

The parser class diagram in Figure 9.17 might look complicated at the first glance, but has a rather simple structure. The core of the parser is formed by

[107] JavaCC (abbreviated for 'Java Compiler Complier') is a lexical analyser and parser generator, available as an Open Source framework at `http://javacc.org/`.

[108] An LL(k) is a top-down parser approach for context-free languages with leftmost-derivation and lookahead functionality for k tokens. (Aho et al., 2006, p. 218)

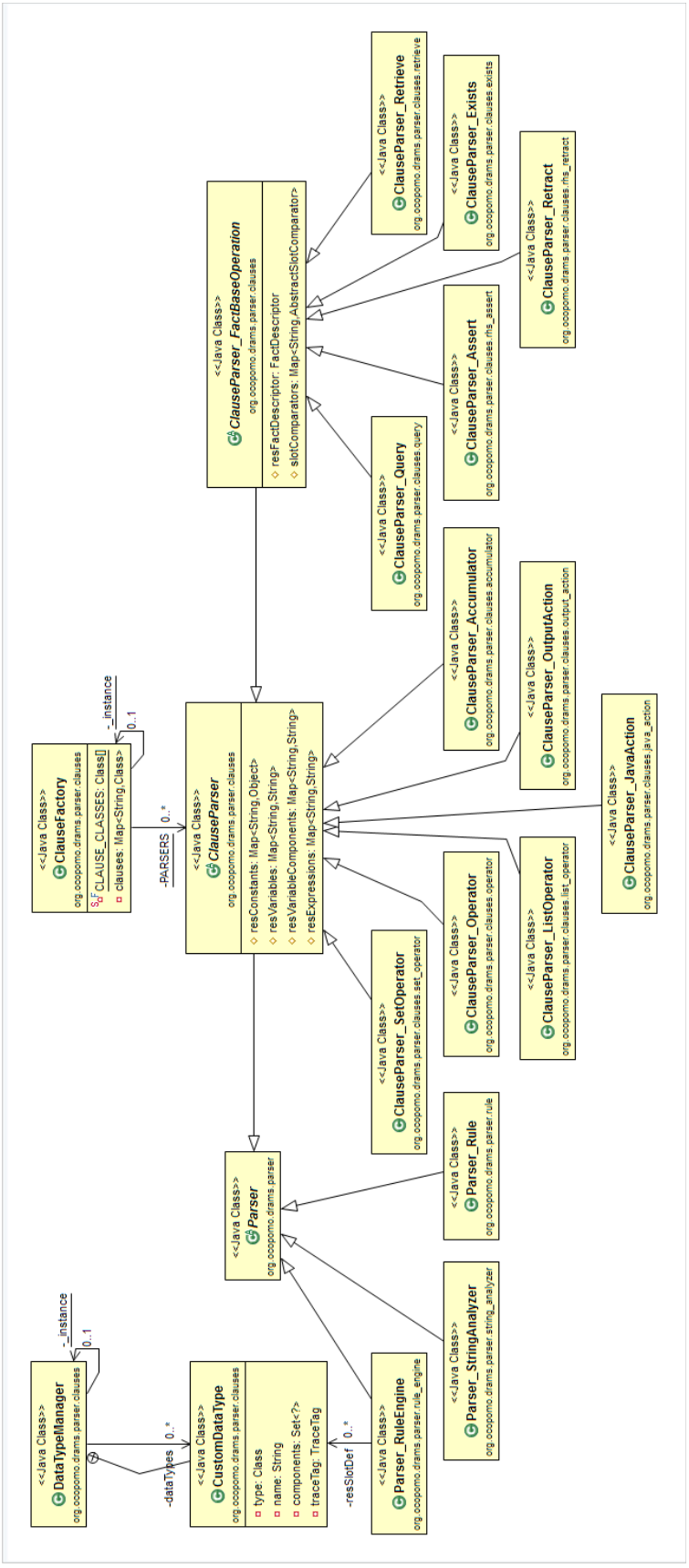

Figure 9.17: Class diagram of parser component

three abstract classes in an inheritance hierarchy: `Parser`, `ClauseParser` and `ClauseParser_FactBaseOperation`.

Concrete implementations of the `Parser` class are:

- The top-level parser `Parser_RuleEngine` allows to parse a DRAMS definition file, create `FactTemplate`, `Fact` and `CustomDataType` objects. The former two have been introduced before, and only the latter is shown in the diagram. This custom data type is comparable to a Java enumeration which specifies a limited set of valid values for a string variable, maintained by the `DataTypeManager`. For rule definitions, the rule engine parser invokes the Rule parser.

- The rule parser (`Parser_Rule`). For the clauses included in a rule, the parser invokes the concrete `ClauseParser` implementations, as described below.

- The special-purpose parser `Parser_StringAnalyzer` for parsing argument strings for Java action clauses.

A specialisation of the `Parser_RuleEngine` is the abstract `ClauseParser`. This is the super class of all clause parser implementations, invoked by the rule parser. A `ClauseFactory` identifies and applies the correct parser for a given code fragment for a clause definition. The concrete implementations are named `ClauseParser_` with a suffix referring to the clause class that is generated by the parser. For example, the `ClauseParser_Operator` creates `Operator` clause objects from the parsed code.

A further specialisation of the `ClauseParser` is the `ClauseParsers_FactBaseOperation`, the super class for all concrete parser implementation for clauses involving fact base operations.

All parsers do not only take care to create the respective clause object, but also assess the meta information attached to the OPS5 language constructs. The most important example here is the traceability information with which fact templates, facts and rules are annotated by the CCD2DRAMS code transformation tool (see Figure 4.7 in section 4.4.2). For such annotations `TraceTag` objects are created by the parsers. This is topic of the next subsection.

9.3.6 Traceability

The traceability component is the most prominent feature of DRAMS. It basically provides means to monitor and profile the rule engine activities, in particular the inference process. Due to the integration with the OCOPOMO toolbox trace information also outside the DRAMS internals can be incorporated. Thus, if the CCD2DRAMS model-to-code transformation tool is used to create the declarative code, annotations are included that allow references to the elements of the conceptual model. Due to the central role of the traceability concept a separate Chapter 10 is dedicated to this topic.

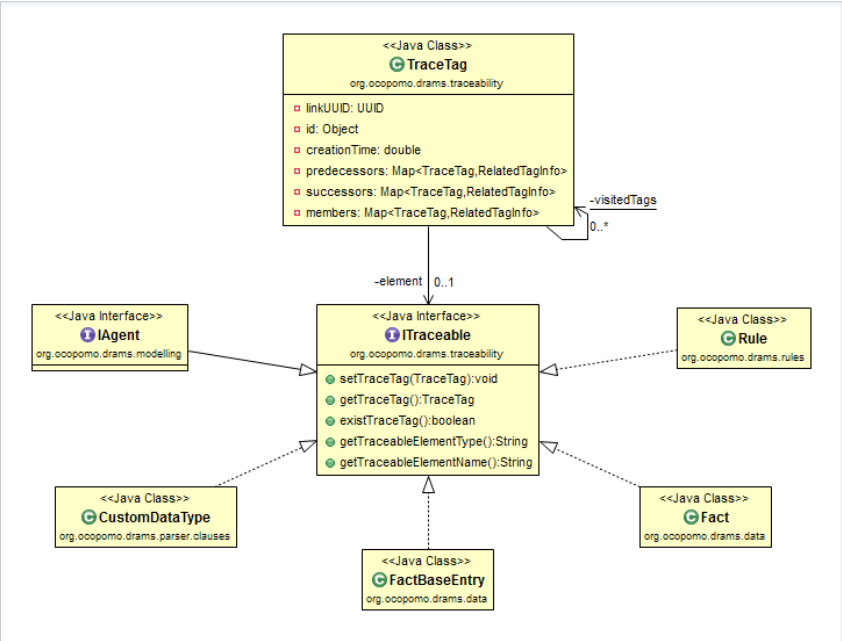

Figure 9.18: Class diagram of trace tag visitor interface

Apart from the coupling to concrete tools, the traceability functionality can also be integrated with other modelling tools since the external trace information is provided by the universal concept of the UUID to maintain a unique reference to any external entity related to a DRAMS construct. Basis for the realisation is the `TraceTag` class in Figure 9.18. This class encapsulates the link UUID in form of the `linkUUID`[109] field, together with the reference to the annotated traceable object in `element`. The trace tags are interlinked with other trace tags (`predecessors`, `successor` and neighbouring `members`) owing to rule engine activities. Hence, an evaluation graph of the various rule firings involved in a simulation run is generated.

In the process of generating the traces, the DRAMS classes `Rule`, `FactTemplate`, `Fact` and `CustomDataType` as well as agents — here represented by the `IAgent` interface — are involved. All these classes implement the `ITraceable` interface which defines several methods for accessing the associated trace tag.

The interaction of `TraceTag` and `ITraceable` is discussed in detail in the following section, so that this concludes the explanation of the DRAMS core components. The following section continues with the interface components.

[109] The link is also captured in a field `id` of type `Object`, so that the link can be of any type. However, (in the current implementation) this results in restricted functionality of the trace tag: such links can only be used to extract textual trace information.

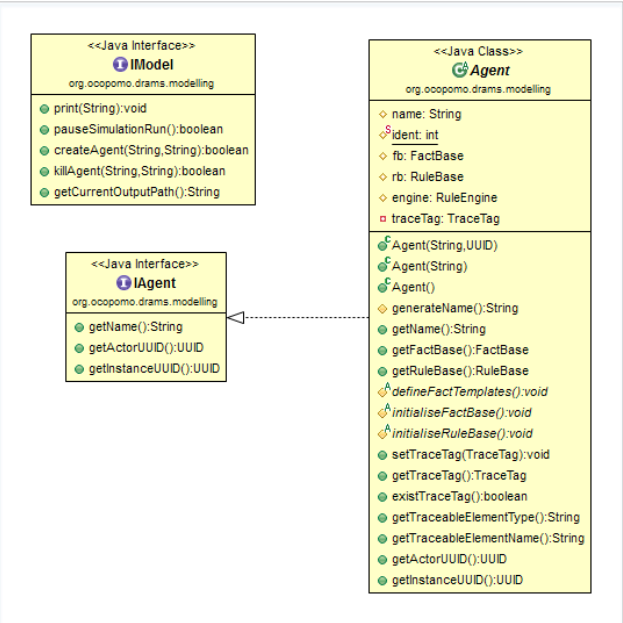

Figure 9.19: Class diagram of simulation model interface

9.4 DRAMS Interface Components

The parts of this section describe the DRAMS API, i.e. the components that make up the means by which DRAMS can be accessed and extended by external software systems. Here only the structure of the API classes should be presented, the functionality and application of these API components is demonstrated together with traceability features in Chapter 10.

9.4.1 Simulation Model Interface

Since DRAMS is tailored to be used with agent-based simulation it brings a package `modelling` with the basic components to be integrated with Java-based simulation models (Figure 9.19). The model itself is intended to provide access to functionality to create new agents, to destroy existing agents, to control (pause) simulation runs and to gain access to logging functionality of the agent model. The respective methods are defined in the `IModel` interface. A Java-based simulation model needs to have this interface implemented by one of its classes. In case of Repast for example, the main model class can cater for this requirement. In turn, the model class implementation is also the appropriate place to trigger the DRAMS rule engine processing every tick (unless DRAMS is used in event-based mode, where the current time is set by the rule engine schedule, not an external timer).

On the other hand, DRAMS brings support for creating agents. As a definition of the minimal required functionality the interface `IAgent` specifies methods to access information (name, universal id etc.) of the agent, while

it specialises the `ITraceable` interface that equips the agent with traceability support (see section 9.3.6 and, for more detailed information, Chapter 10). In addition, DRAMS comes with an abstract `Agent` template class which implements the full integration of DRAMS in an agent, i.e. the Java access to the individual rule engine with fact and rule bases and the `TraceTag` integration. A concrete agent implementation just has to override the abstract methods `defineFactTemplates()`, `initialiseFactBase()` and `initialiseRuleBase()` for Java-based initialising of the rule engine (though all these initialisations can also be done via the DRAMS language/parser).

9.4.2 Plugin Interface

The architecture of DRAMS is designed in a modular way, focussed on core functionality. For example, there is no graphical user interface integral part of DRAMS, nor are components to write simulation outcomes to different formats and files. Not having such functionality integrated makes DRAMS more easily customisable, without the need to interfere with the core component code. In this context, customisation refers to extending the functionality of DRAMS for dedicated needs, e.g. to allow the creation of specific output file formats or to add special graphical visualisations or user interfaces.

For this purpose DRAMS is equipped with a plugin interface which consists of a number of Java interfaces and a `PluginManager` for loading plugins and establishing the necessary data connections to the DRAMS core. An overview is given in Figure 9.20. The interfaces have the role to provide information about the plugin to DRAMS.

At least two interfaces need to be implemented by each plugin project. On the one hand, each plugin must offer DRAMS a description about its basic properties, on the other hand the functionality which is extended by the plugin needs to be made known to DRAMS. The former is done via the `IPluginDescriptor` interface. It defines methods to request information about the plugin name and version numbers, and — most importantly and referring to the latter point above — about the extension points. The extension points are basically a collection of implementations of the `IExtensionDescriptor` interface, more concretely of one of the specialisations of this interface:

- The `IUIExtensionConnector` is a general marker interface for plugins providing UI extensions for DRAMS. A specialisation of this interface is the

- `ISwingUIExtensionConnector` for Swing[110]-based user interface extension. A user interface extension is typically an implementation based around a main window[111] which the plugin makes accessible to DRAMS

[110] Swing is a graphical user interface toolkit and part of the Java Foundation Classes, provided by the Oracle Corp.; `http://docs.oracle.com/javase/tutorial/uiswing/start/about.html` (retrieved on 2017-04-20).

[111] In Swing-terms called Frame.

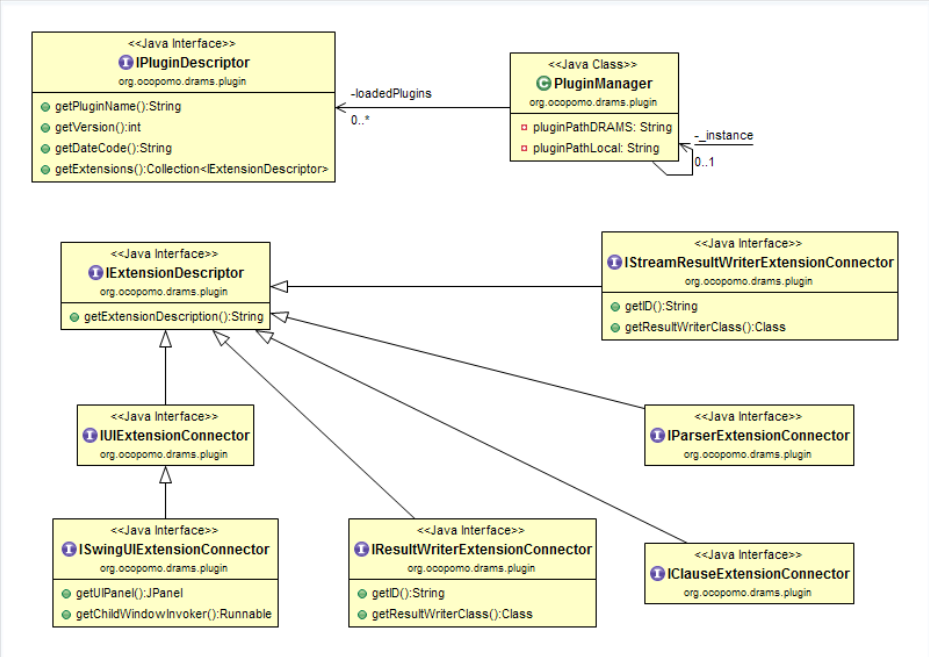

Figure 9.20: Class diagram of plugin interface

via the method `getUIPanel()`. DRAMS takes care to instantiate the window and other plugin elements on start-up using the `Runnable` (Java thread) object retrieved by the interface method `getChildWindowInvoker()`. The UI extension is relying on the Java-realisation of the Model-View-Controller (MVC) design pattern: The DRAMS core classes that want to show their status to plugins implement the `Observable` interface, the plugins in turn implement the `Observer` class.

- The `IResultWriterExtensionConnector` is implemented by plugins that provide result writer functionality. Result Writers need to register at DRAMS with an identifying name, which is used in the DRAMS language to specify by which Result Writers (i.e. in which output format) simulation outcomes are further processed or stored. These plugins also need to implement a class that extends the `AbstractResultsWriter` classes supplied by DRAMS (see next section 9.4.3) which is able to receive output data produced by DRAMS. A reference to this class object needs to be provided via the `getResultWriterClass()` method.

- `IStreamResultWriterExtensionConnector` is a similar interface `IResultWriterExtensionConnector` for plugins that want to receive output data via Java streams, hence drawing upon the `AbstractStreamResultWriter` class.

Two more extension descriptors are foreseen and incorporated in the DRAMS core, but not fully implemented yet:

- With the `IClauseExtensionConnector` new clauses can be defined, extending the capabilities of DRAMS rules and the extent of the DRAMS language. Such plugins must provide a clause class, a list of keywords for the parser and a ClauseParser-subclass for implementing parsing functionality.
- The `IParserExtensionConnector` can be used to extend the DRAMS language at levels above clause level. This could be used to define a complete new language for fact template, fact and rule definitions, e.g. a graphical definition language.

The way by which DRAMS does the binding with plugins is dependent on a correct implementation of the `IPluginDescriptor` interface. The implementing class within the plugin must have the obligatory name `PluginDescriptor` in order to have a defined entry point of the plugin for the `PluginManager`. This singleton component loads the plugins in the following regime:

- Plugins are deployed as Java Archives (so-called JAR files). During initialisation of DRAMS, the manager searches in several distinct folders for DRAMS plugin .jar files.
- For any plugin found, the latest version available in the folder is automatically loaded and initialised.

A concrete application example of the plugin interface is demonstrated in the next chapter, section 10.2.

9.4.3 Result Writer

The DRAMS Result Writer is the facility to collect outcomes during rule engine processing and forward the cumulated data to output writer plugins for further processing or storage. The main entities of the Result Writer component are the `AbstractResultsWriter` class and the sub-class `AbstractStreamResultsWriter`, as shown in the overview class diagram in Figure 9.21. Each Result Writer plugin has to extend one of these two classes.

The main functionality for collecting result data from rules in a structured way is encapsulated in the `AbstractResultsWriter` class. Rules can produce textual logs (via print clauses), but also any kind of numerical data, both on a cumulated level as well as individual for each agent. Accordingly, it contains various member variables, besides the name and various flags for processing options (the Boolean variables), most importantly a table structure to collect and pre-process the results produced by rules. In this context the columns of the table are called facets. The `facets` list stores the possible column headers of the table, which are the names of the result item in the cumulated mode, and result item name concatenated with the name of the agent instance that generated the result when used in individual mode. There are also member variables for

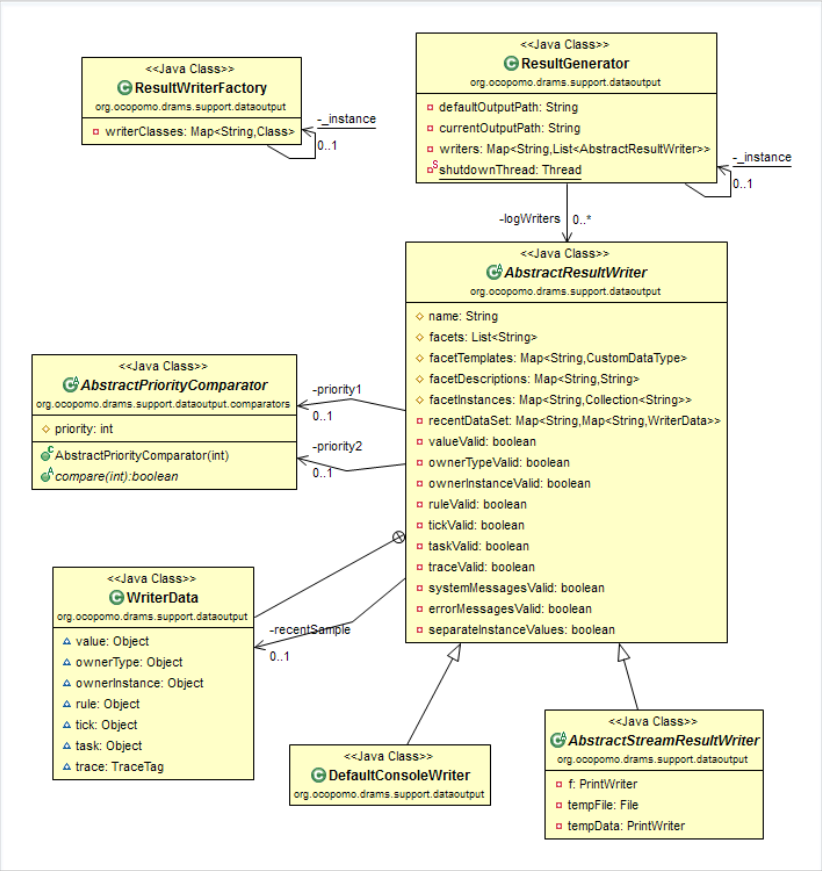

Figure 9.21: Class diagram of Result Writer

- `facetDescriptions` which contain descriptions of the facets and can be seen as part of the facet name;
- `facetTemplates` which define the data types of the facets, expressed by the `CustomDataType` class;
- `facetInstances` which is a map filled during rule engine processing with the agent instances which sampled data to the facet. The instance names are the basis for the agent-individual mode.

The actual table data structure is represented by the `recentDataSet`, a two-dimensional map with `WriterData` as content elements and result name and generating instance as keys. `WriterData` not only contains the actual data item, but also meta data like information about the owner of the rule producing the output, the rule itself, the generation time as well as a (possibly existing) trace tag.

Additionally, priority information can be specified for a Result Writer. This is expressed by two `AbstractPriorityComparator`s. With these priority values results can be directed to different output writer plugins, e.g. to distinguish actual simulation results from debug outputs.

`AbstractStreamResultsWriter` is a sub-class of `AbstractResultsWriter`, specialised for routing the results to a Java `PrintWriter` output stream. This class provides functionality to make the result samples persistent.

The `ResultGenerator` singleton class manages the Result Writers and controls the output data flow. All concrete Result Writers (i.e. the respective plugins) have to be registered there. Main functionality is to determine the destination folder for persistent result data (result files) and to take care that no data is lost in case of 'abnormal' program interruption (e.g. by user interaction).

Figure 9.21 also shows the only concrete Result Writer implemented as part of DRAMS, the `DefaultConsoleWriter` for writing data to the default output device, e.g. logging capabilities of the model implementation or the default Java output stream.

Finally, the `ResultWriterFactory` is part of the Result Writer component, a singleton factory class to instantiate concrete fact writers from a given class name set by the Result Writer plugins.

9.4.4 UI Manager

The UI manager component is the connection point for extending DRAMS by (graphical) user interface components, which are also implemented as plugins. In contrast to the output writer component introduced in the previous section where the data produced by rules is regarded, here views on internal DRAMS data structures and status information are in focus. Thereby three different UI views are distinguished, defined by interfaces extending the marker interface `IView`, according to Figure 9.22:

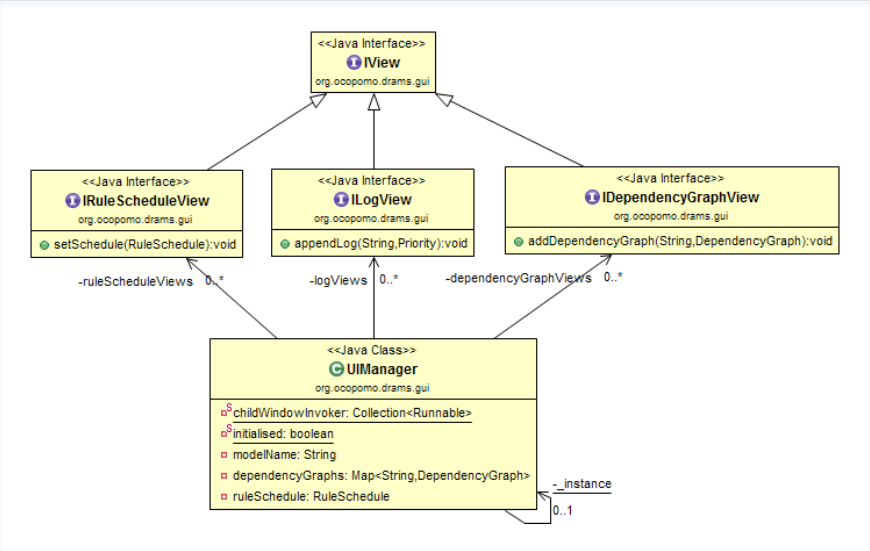

Figure 9.22: Class diagram of UI manager

- The `IRuleScheduleView` interface has to be implemented by UI plugins that include a visualisation of the rule scheduler log. This log contains status information about the rule schedule, e.g. information on which rules are scheduled at which time and for which agent instance. The method `setSchedule()` is called by DRAMS UI manager to hand over a new rule schedule to the plugin.

- The `IDependencyGraphView` interface allows UI plugins to gain access to the dependency graph's data structure in order to provide a respective visualisations. The method `addDependencyGraph()` is called by DRAMS UI manager to forward a `DependencyGraph` object to the viewer plugin.

- The `ILogView` interface finally is implemented by UI plugins that allow to view the DRAMS system log. Depending on the log level, warnings and error messages are passed to the plugin via the `appendLog()` method.

The `UIManager` singleton class manages the UI plugins, which have to register their invoker thread object (using the method `addAWTWindowInvoker()`) as well as all implemented viewers (by the `addView()` method) at the UI manager. The interface methods `setSchedule()`, `appendLog()` and `addDependencyGraph()` are invoked by DRAMS during rule engine activities, if new respective elements are available.

9.5 Conclusions

This chapter provides detailed specifications of static and dynamic aspects of DRAMS, and can serve as a guide to the source code. Further documentation

is attached with the source code to be accessed via the online resources in appendix C.2.

Although DRAMS is used in several simulation projects, several improvements and extensions are conceivable:

- The currently exclusive way to write DRAMS code is to use the OPS5-like language. However, the parser for this language is implemented in a modular way with well-defined interfaces, so it would be possible to replace it for a different language without too much effort. An interesting and certainly valuable replacement language would be some kind of graphical design language, for example similar to the DDGs presented in Chapter 6.

- Another major construction area regards measures to improve the execution speed and lower the memory consumption of DRAMS in some cases. DRAMS uses in-memory fact bases, which make extensive fact bases operations quite quick, at the cost of increased use of memory. Speed could be improved by a new scheduling algorithm and caching, taking not only LHSs into account, but relying on individual clauses.

Chapter 10

Implementing traceability in DRAMS

10.1 Introduction

This chapter finally gives the technical view on the realisation of traceability, the encompassing concept throughout this PhD thesis. Although the aspects shared in this chapter are linked to DRAMS, the general concepts also apply to other environments. This chapter aims to provide an example how the general traceability concept defined in section 2.2.5 can be implemented, in order to support simulation result analysis as shown in Chapter 7, and thereby 'putting flesh on the bones' of the modelling process specified in Chapter 3, from an implementer's point of view. The content of this chapter in unpublished original work.

The chapter is structured in two sections: Firstly, the prerequisites and mechanisms for trace generation are described as means for realising traceability. Secondly, it is shown how traceability information can be exploited from a technical perspective. In this context the DRAMS plugin API as well as a specific plugin for visualising traces (the Model Explorer) are characterised.

10.2 Realising Traceability

As mentioned in section 8.3.2, generating evidence traces of simulation runs is one of the main requirements for the development of DRAMS. Motivation for generating traces of a simulation process is most prominently driven by the desire to augment simulation results with information about evidences the simulation model is based upon. In contrast to theory-driven simulation models where the model is verified against e.g. a social theory, evidence-driven simulation models often use cognitive heuristics for abstractions made for important model topics found in evidence. Here evidence is defined as any kind of textual material, like scenarios (e.g. created by the scenario method), publically available background information (e.g. newspaper texts), but also application field

specific material ('domain documents') like police interrogation protocols as basis for a simulation model of a criminal network. For this kind of modelling problem, in the OCOPOMO project the aforementioned modelling method has been developed, together with an accompanying toolkit of which DRAMS is one of the building blocks.

A traceability concept as a possible 'golden thread' through both a modelling process and a toolbox has been introduced in section 3.4. Figure 3.13 illustrates how the different artefacts are bonded together: Starting from the documents of the evidence base, the consistent conceptual description (CCD) represents the first stage of abstraction, still in a structure and language easily understandable for application field experts and non-modelling application field experts. The formal simulation model then preserves links to the CCD elements, and attaches trace anchors in form of hyperlinks to the simulation results. In the presentation of simulation results the links can be replaced with the text phrase attached to the CCD elements.

The following subsections will elaborate the technical realisation of this traceability concept in DRAMS. Starting with the prerequisite of model code annotations, a detailed description of the mechanisms for processing this trace information within DRAMS is given.

10.2.1 Annotating Model Code

In the track of passing link information from provenance data to simulation results, the pathway between model code and raw simulation outcomes can become quite complicated. This is primarily due to the fact that the means for processing initial data configurations within a simulation run can be very complex, and are influenced by manifold factors. When speaking in terms of agent-based declarative simulation models, each 'individual' in an agent population of arbitrary size carries up to thousands of facts, which are processed by hundreds of rules. Under such conditions adequate methodological and tool support is mandatory to enable the extraction of useful information for understanding structure and behaviour of the model.

A methodological approach for solving this issue has been discussed in Chapter I: a conceptual model, here the CCD, is developed prior to programming the simulation model as part of a modelling process. This CCD incorporates and specifies all the crucial and evidence-related elements for the simulation model, but abstracts from technical necessities for making a simulation model 'run'. Hence, only those simulation model elements are incorporated in the traces which have counterparts in the CCD.

As a starting point, these crucial elements have to be marked in the simulation model code by link annotations. Each element in the CCD is equipped with a unique identifier (UUID), and this link is attached to the related simulation model code element (as shown in Figure 10.1). For the OCOPOMO toolbox, these link annotations are maintained by the model-to-text code gen-

```
/*aggressiveAction -> Criminal
 *@link _EA9M8BKpEeaGA-T5FOUApg */
(deftemplate
BlackCollarCriminal::R_appliesTo_Criminal(aggressiveAction:String)(Criminal:String))


/*Action: member X decides to betray criminal organisation
 *@link _56Z-ECnjEeOEY6M6x66YnQ

 * Annotation in "File_research_Gloders_D.txt": " Am x hat die Criminal Intelligence Unit der
regionalen Polizei x erhielt einen anonymen Brief."
 * Expert Annotation by "": "On x the criminal intelligence of x received an anonymous letter.
"
 */
(defrule BlackCollarCriminal::"member X decides to betray criminal organisation"
//(::normEvaluation(normDemanded invalid))
//TODO
=>
//TODO
)
```

Figure 10.1: Generated DRAMS code with a fact template definition (deftemplate) and a rule stub (defrule) with UUID link annotations (@link)

eration tool CCD2DRAMS (Scherer et al., 2013b). This code generator adds link annotations to all generated elements, in particular for agent classes and instances, fact templates, facts and rule stubs.

The model programming then consists mainly in creating the code around the generated parts, on the one hand by filling in the complete intended logic into the rule stubs, on the other hand by providing 'glue' code in-between the generated code. The latter consists of code to

- perform fact base operations,
- generate (numerical or textual) model outcomes,
- join post-conditions from rule firings and prepare appropriate pre-conditions for other rules, and
- reflect constraints set by model assumptions not reflected in the evidence.

Such development approaches are usually not sequential but rather cyclic. This means that during model programming missing crucial elements are discovered for which evidence can be found. These have then to be added at the CCD level. The CCD2DRAMS code generator takes care not to overwrite already elaborated rules — situated in a user code section of the source file — when re-transforming a modified CCD. During parsing of model code by DRAMS, for each element with a link annotation a so called trace tag is generated. This is basically a small data container storing the link UUID, together with the possibility to define different kinds of neighbour trace tags. These are used as nodes in the generated trace graph, as shown in the next section.

10.2.2 Creating the Traces

In DRAMS, the creation of trace information is an ancillary procedure of the forward-chaining rule engine process. The necessary extensions and involved algorithms are explained in this section. Later on, a view on static aspects, including the involved classes and data structures, is given.

Dynamic aspects of trace generation

Generating simulation trace information is basically the process of binding the link UUIDs to simulation outcomes during rule engine processing. The starting point (or necessary pre-condition) is to forward the UUIDs attached to CCD elements to the DRAMS core objects. This is achieved by evaluating the DRAMS code annotations with link UUIDs by the parser component. The UUIDs are internally stored in so-called trace tags (i.e. objects of the class `TraceTag`), which are attached to fact template, fact and rule objects.

A detailed picture of the involved data elements and Java objects is given in Figure 10.2. Several perspectives are combined in this figure. On the horizontal axis the three vertical background boxes outline the sequence of rule processing: the pre-condition in the left-hand box, the rule processing in the middle box and the resulting post-condition in the right hand box. On the Y-axis the objects involved in rule processing are shown in three different layers, shown as horizontal background boxes. In the top box the data-rule dependency graph from the example previously introduced in Figure 6.2 is used again to represent the pre-condition fact, the dependent rule and the generated post-condition fact[112]. These are the objects derived from related CCD elements, thanks to the automated model-code transformation automatically attached with the UUID link (represented in the figure by dashed curved arrows). The DRAMS-internal representation of these three objects is shown in the box below. Each of the two facts is represented by a Java `Fact` object, attended by a related `FactBaseEntry` object, i.e. the fact template. For the rule a corresponding `Rule` object exists. This object itself is a composition of clauses, from which the fact base retrieve (for the LHS) and the fact base assert (for the RHS) clauses are relevant for the trace generation. The traces are generated by creating `TraceTag` objects during the rule processing, linking the `FactBaseEntry`, the `Fact`, the `Rule` and the clause objects together. The respective example of this trace graph data structure is depicted in the bottom background box of Figure 10.2, with the solid vertical curved arrows pointing to the associated DRAMS core objects.

The trace tag objects referring to the different kinds of DRAMS core objects are all using the same `TraceTag` class. Besides the actual reference element, two member variables are pointed out: The `linkUUID` referencing to an external entity — here a CCD element — and the `creationTime`. The

[112]For simplification single facts are shown for pre- and post-condition. In a real model an arbitrary number of both kinds of facts may appear.

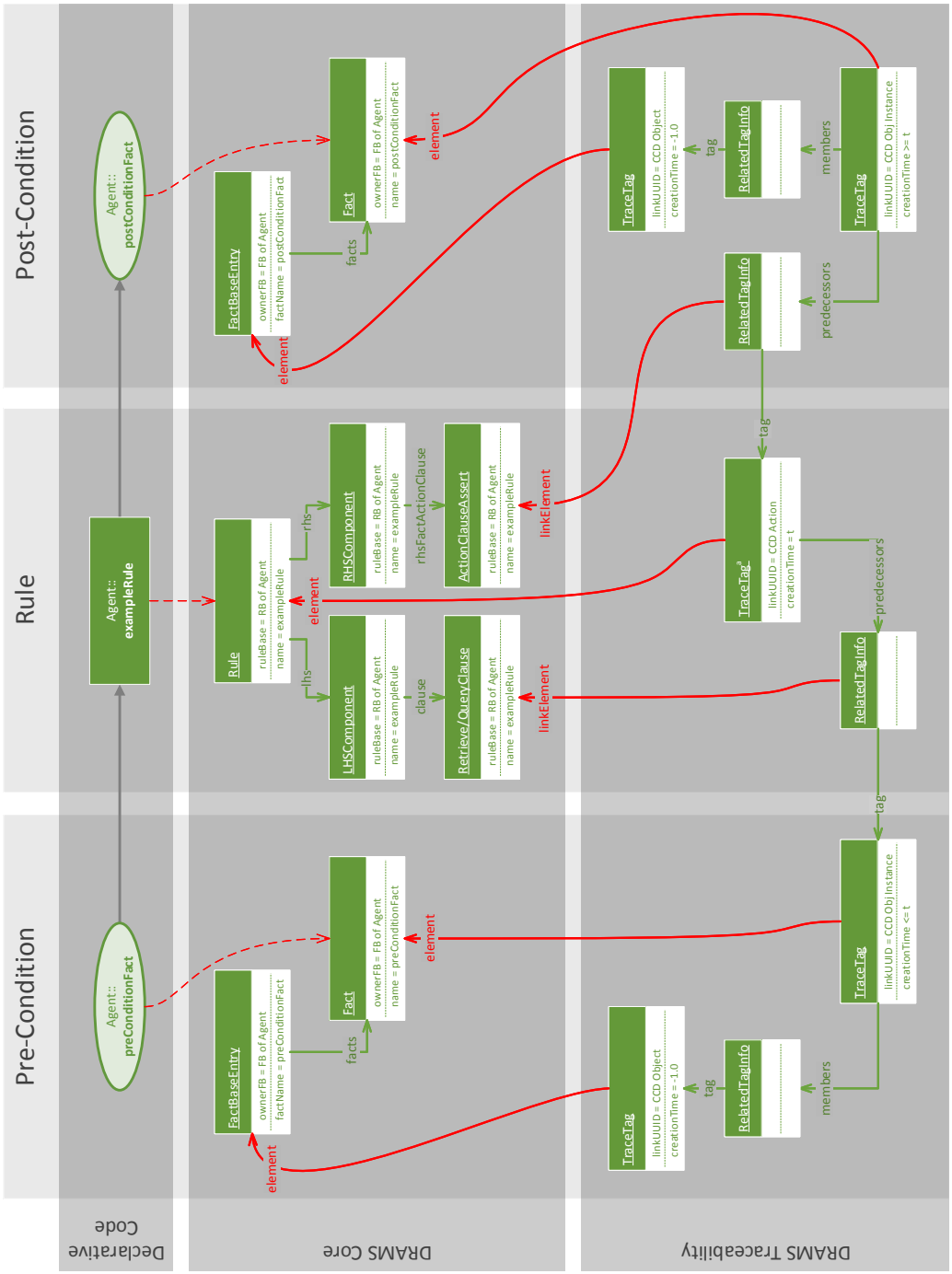

Figure 10.2: Interrelations between objects of different DRAMS layers. Remark (a): linkUUID and element derived from initial rule trace tag, generated at creationTime = -1.0 (not shown in the diagram).

`creationTime` of the trace tags is dependent on the type of associated core element and differs for the included trace tags. For trace tags associated to fact templates or facts the creation time is determinative. Fact templates (i.e. the `FactBaseEntry` objects) are only instantiated at rule engine initialisation, thus a creation time of -1.0 is set, a value that has been arbitrarily selected to represent the time before the distinct starting point of (simulation) time (i.e. the 'undefined past'). Pre-condition facts might be instantiated at any time before being evaluated by the LHS, hence the noted creation time has to be less or equal to t (depending on the specified lag[113] mode of the retrieve operation), the time of rule evaluation and possible firing. In the trace tag associated with the rule, t is noted as creation time. This trace tag is not the actual rule trace tag (also instantiated at rule engine initialisation, not shown in the figure), but a clone of this object dedicated for the current rule firing. The trace tag for the post-condition fact has to be instantiated after the rule firing, hence the creation time must be greater or equal to t (similar to the lag mode for fact base retrievals a fact assertion can be deferred by a specified number of ticks).

Two types of relations between `TraceTag` objects are incorporated in the diagram[114]. Between trace tags for facts and rules the predecessor relation is used, pointing out the temporal flow of the inference process, resulting in an activity graph reflecting the involved rule engine processes. The second type of relation between trace tags is the member relation. In the example it is used to keep track of the fact templates related to an actual fact. Another member relation would be the owner agent of a fact or rule.

Predecessor and member links do not point directly to the related `TraceTag` objects, but rather to objects of the class `RelatedTagInfo`. This class allows to add additional information for links, labelled as `linkElement`. Here in the example these are used to record information about the actual retrieve or assert clauses related to the predecessor relation, i.e. pointing to clauses which actually retrieved or asserted relevant facts. This information serves for example to highlight relevant parts of rule code, which will be illustrated in the following section.

Summarising, the `TraceTag`s are found in the following roles:

- The predecessor trace tag
  - of a fact is associated with the rule that caused the existence of the fact;
  - of a rule is associated with the facts that triggered a successful LHS evaluation.

[113] A lag mode specified temporal criteria for the fact to be considered in a fact base retrieval. For example, with the lag mode 'last' only facts that have been asserted in the last tick are retrieved.

[114] In fact three types of relations are maintained in the implementation: an additional successor link is included, mainly for technical reasons, e.g. to perform efficient graph algorithms for analysis and visualisation purposes.

- The successor trace tag
  - of a fact is associated with the rules fired due to the existence of the fact;
  - of a rule is associated with the facts for which it was responsible for the assertion.
- The member trace tags can be associated
  - for a fact, with the fact template and the owner agent;
  - for a fact template and a rule, with the owner agent, always provided that these elements have a trace tag.

The process of creating predecessors and successors can be outlined as follows. During rule processing (i.e. when a new rule fires or when a new fact is asserted), the trace tag of this (new) element is extended by information about the predecessor. E.g. for a rule, the facts that caused the successful evaluation are linked, and for a newly asserted fact the rule is linked which asserted the fact. At the same time, at the trace tag of the preceding element is extended by a successor, namely the new rule or fact, respectively.

The simplified algorithm for trace generation (as outlined in Figure 10.2 and initially published in Lotzmann and Wimmer (2013a)) is comprised of the following steps:

1. At the initial state of the rule engine — no rule has fired — a number of fact templates, partly concrete facts for the templates and rules are present. For subsets of each of these elements (for which CCD elements exist), trace tags are attached.

2. When the rule engine is initiated, it firstly checks which rules might fire with the given set of facts. The LHS's of these rules are evaluated, and for each successful evaluation the RHS is triggered.

3. The RHS processing starts with checking whether at least one of the facts evaluated by the LHS (and, hence, determining the data basis for the RHS execution) is attached with a trace tag. If this is the case, a new trace tag for this particular rule firing at the current simulation time is generated using the information (link UUID) stored in the rule trace tag, if available. All trace tags for the LHS facts are then incorporated as predecessors of the new rule firing trace tag.

4. The rule firing trace tag, or the rule trace tag, according to disposability, is then passed to all RHS clauses.

   (a) A clause for asserting a new fact to a fact base generates a new trace tag for this fact with the trace tag delivered by the rule as predecessor.

(b) A clause for writing output data (e.g. a log record) passes the trace tag to the Result Writer DRAMS plugins.

5. When all rules have fired, the newly created facts constitute the new state of the rule engine, and the processing continues with step 2.

Step 4.(b) in the algorithm generates so-called connector trace tags. These are root nodes of directed acyclic graphs which cover all relevant elements and processes that created the simulation outputs (e.g. log entries). A simulation run produces a potentially very large graph data structure, holding information about traces for all relevant generated facts and simulation outcomes. With this kind of meta-data for the simulation run, causes of particular simulation results can be analysed quite easily, applying various graph algorithms and visualisation (see section 10.3).

Class structure and static relations for supporting trace generation

From a static perspective, the reference-`linkUUID` to CCD elements is encapsulated in the `TraceTag` class as shown in Figure 10.3. This class realises the so called *trace tag*, the DRAMS concept for attaching link information UUIDs to types, fact templates, facts, rules and other entities like agents. With the traceability approach of DRAMS the trace tag is the node element in the evaluation graph, basically a tree-like graph data structure reflecting the steps and results of an inference process starting from an arbitrary initial configuration of the rule engine. The term tree-like refers to the fact that although the inference process including facts and rules is represented by a real tree, relations to 'neighbouring' elements such as fact template and owner agent are also included, hence edges to elements of the same tree level may occur. This evaluation graph is to some extent comparable with *Retes* (mainly the Alpha network), the directed acyclic graphs applied in *Rete* algorithms (Forgy, 1982). In the context of DRAMS they are not applied for pattern matching (this is done by the data-driven dependency graph approach introduced in the two previous chapters), but rather used to reconstruct and visualise the inference process, and at the same time to extract the traceability information from the individual activities. An example how both kinds of information can be combined is given with the Model Explorer plugin for DRAMS, introduced in subsection 7.3.1 and elaborated further in the following section 10.3.

All classes for which trace tags can be attached implement the `ITraceable` interface. As indicated in Figure 10.3 these are `FactBaseEntry` (here referring to a `CustomDataType`), `Fact`, `Rule` and the interface `IAgent`, which in turn is implemented by the `Agent` template class provided by DRAMS.

These trace tag objects are generated in association with the tagged element, for rules by the `Parser_RuleEngine`, for fact templates, facts and custom data types by `Parser_RuleEngine` (for elements defined in the DRAMS code) as well as by the `ActionClauseAssert` (for facts asserted by clauses).

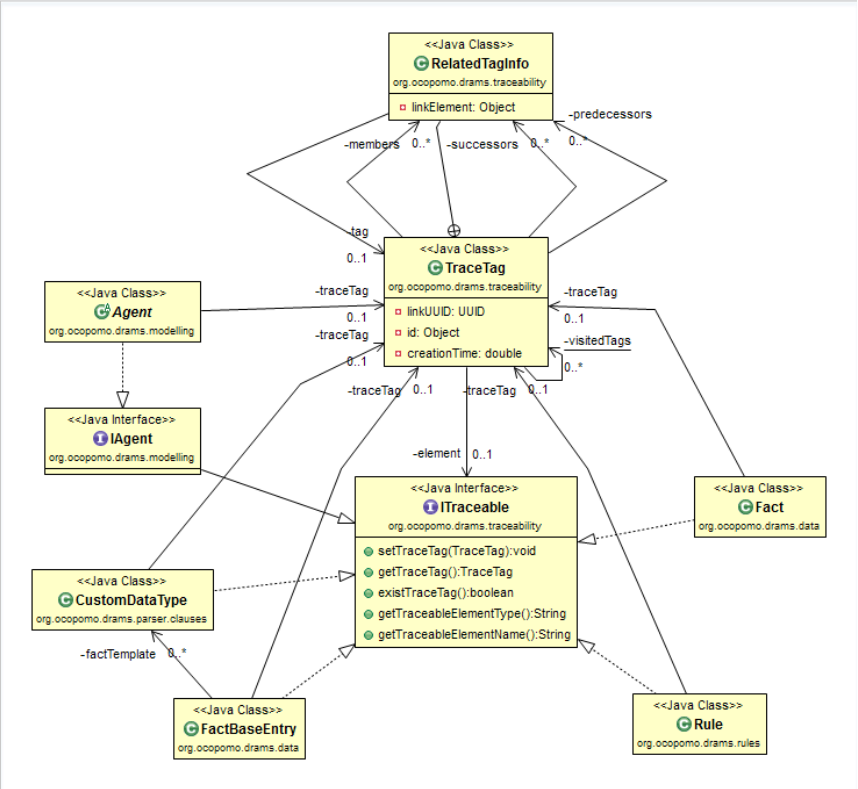

Figure 10.3: Class diagram of traceability component

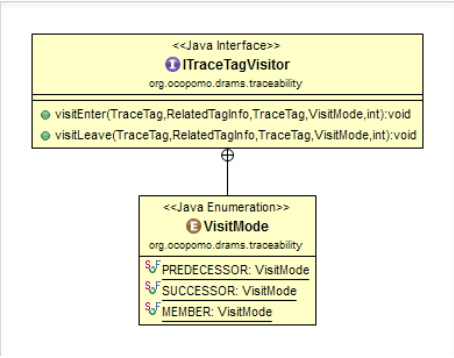

Figure 10.4: Class diagram of trace tag visitor interface

In order to build an inference graph during rule engine processing, a trace tag has to maintain relations to adjacent trace tags. As the same trace tag class is used for each traceable element, the adjacent trace tags belong to different kinds of elements. For example, the trace tag of a rule collects trace tags of all facts that caused the firing of the rule, and all facts asserted by the rule (each with a reference to the clause in charge for the fact base operation; see below). The adjacent trace tags are classified in three categories: predecessor, successors and members. All these linked elements are represented by the `RelatedTagInfo` class.

This inference graph can be traversed by visitor objects in both directions (i.e. the links are designed to be bi-directional). For this purpose the `TraceTagVisitor` interface has been introduced (Figure 10.4).

A standard approach following the visitor design pattern (Gamma et al., 1995) is envisaged here. Classes implementing this interface have to implement (at least one of) the two methods `visitEnter()` and `visitLeave()`. According to the graph data stucture to be traversed, three visiting modes for predecessor ('backward'), successor ('forward') and member ('sideways') are specified in the `VisitMode` enumeration. The Model Explorer introduced in the next chapter gives an implementation example for this visitor interface.

10.3 Exploiting tracebility information

This section is mainly dedicated to the DRAMS Model Explorer, a tool that allows the graphical visualisation and inspection of the rule evaluation graph generated by DRAMS. It has been developed in the OCOPOMO project as a plugin for DRAMS, and initially been discussed in (Lotzmann and Wimmer, 2013b) and (Lotzmann and Wimmer, 2013a).

The Model Explorer has already been introduced from a user perspective in section 7.3.1. In the following subsections a technical perspective on this tool is provided, including architecture and implementation aspects. Firstly, an overview is given on the different options to process simulation outcomes

by DRAMS plugins.

10.3.1 Processing Simulation Outcomes

DRAMS brings a plug-in interface by which any kind of output processing facilities can be embedded. Such plug-ins can either write files of a particular format, or can serve as an adapter to — for example — an analysis or visualisation tool.

A selection of implemented plug-ins for DRAMS is listed in the following compilation, each representing the trace information in different ways and formats:

- Plain Text / CSV — these two plug-ins write log records or numerical outcomes in plain text files or CSV tables. In both cases, the trace information can be attached as lists of UUIDs, optionally attached with additional information (e.g. name of the element belonging to the UUID). For these formats, the usage of the UUID is usually restricted to manual handling.

- XML — this plug-in creates XML files containing numerical or textual simulation outcomes. These XML files are processed by another component of the OCOPOMO toolbox[115] for creating traceable logs or different types of diagrams. The trace information can be added to the values in different levels of details, e.g. as a simple collection of UUIDs (as for text output above), a diary with the involved UUIDs for the different simulation time steps, or a complete XML representation of the evaluation graph.

- Model Explorer Tool — this plug-in provides an UI for displaying the simulation log, and — by selecting a log entry — the related trace information is visualized and can be analysed in various ways. This tool is subject matter of the remainder of this section.

10.3.2 Architecture and realisation of the Model Explorer

The technical realisation of the Model Explorer encompasses three main components: The implementation of the interface to DRAMS, a facility for representing and processing the evaluation graph data structure, and the user interface. The architecture and working principle of these three components are outlined in the following subsections.

[115] Alfresco collaboration space — the model-based scenarios presented there are equipped with annotation to the evidence base (see Figure 3.14).

DRAMS Plugin Realisation

The DRAMS plugin API is illustrated in the following in view of two purposes: on the one hand to show how the Model Explorer technically accesses simulation data generated by DRAMS in order to visualise traceability, on the other hand it should also serve as a more general example how to implement the DRAMS plugin interface.

The DRAMS plugin API consists of three parts, each designated with a set of Java interfaces predefined by DRAMS:

- An interface to provide meta-information about the plugin,
- the simulation data exchange interface, and
- the connector to various extension points of DRAMS.

The first part is dedicated to exchange meta information about the plugin, i.e. the name, purpose and version of the plugin. This information is needed to correctly load and initialise the plugin by the DRAMS `PluginManager`. Figure 10.5 shows the relation between the respective DRAMS interface `IPluginDescriptor` and related plugin classes: `PluginDescriptor` implements this interface and provides the meta information, while it also maintains a relation to the `ComponentConnector` class. The `ComponentConnector` is the entry point to the functional parts of the plugin, in this case the UI (shown as member variable of type `TraceOutputWindow`, explained in Figure 10.7) and the data exchange interface implementation, shown by the relation to the class `TraceConsoleWriter`. The `TraceConsoleWriter` extends the `AbstractResultWriter` provided by DRAMS, a class belonging to the second part of the API that enables access to data generated by DRAMS Result Writers, i.e. the log entries of simulation runs. The `BufferEntry` is used to store and preprocess the log entries within the plugin, and serves as the model (in the sense of the MVC design pattern) for the log view of the UI (reference to `TraceOutpuWindow` class). The buffer entries are also considered as entry point to the evaluation graph.

Figure 10.6 shows the third part of the DRAMS plugin API, the connector to DRAMS extension points. For the Model Explorer, two extension points are implemented:

- The `IResultWriterExtensionConnector`, a DRAMS interface that allows to add new Result Writers. DRAMS supplies these Result Writers with log entries produces by rules (by means of *write* or *print* clauses). The class `ResultWriterExtensionConnector` implements this interface within the plugin.
- With the `ISwingUIExtensionConnector` the (Swing-based) UI part of the plugin is seamlessly integrated in the DRAMS UI. This means that DRAMS initialises and displays the Swing frame (i.e. the UI window) of the plugin at start-up.

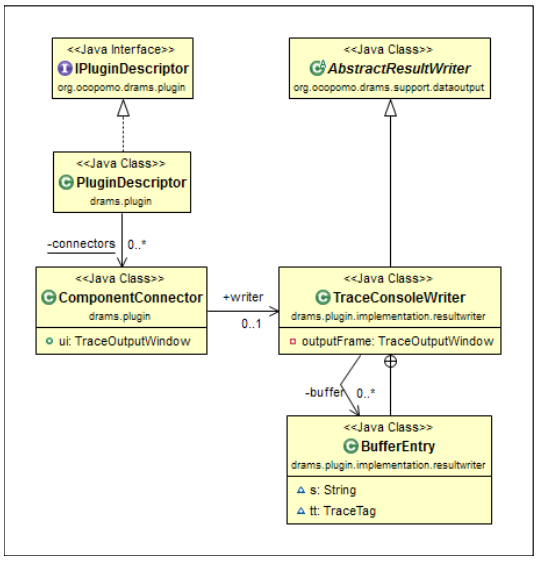

Figure 10.5: Class diagram of plugin descriptor

Figure 10.6 additionally shows the hierarchy of DRAMS extension connector interfaces used by the Model Explorer, all derived from the super interface `IExtensionDescriptor` (a complete overview of all the provided extension point interfaces can be found in Figure 9.20). A plugin can connect to the desired extension point simply by implementing the respective interface — in this case by the classes `ResultWriterExtensionConnector` and `UIExtensionConnector`.

The implementation of the `ResultWriterExtensionConnector` enables access to the Result Writer implementation of the plugin (class `TraceConsoleWriter`) to DRAMS. Its integration within the UI part of the plugin — based on the `UIExtensionConnector` — is sketched in the following subsection.

User interface component

The Model Explorer UI is constituted by two classes: `TraceOutputWindow` realising the main part of the user interface, and a dialogue box `TraceOutputDialog` for providing traceability and rule implementation details. From a perspective of the MVC design pattern, these two classes as shown in Figure 10.7 combine both the view and the control, together with the model part represented by the `TraceConsoleWriter`.

The UI classes are based on the Java Swing framework. While the `TraceOutputDialog` just contains UI functionality, the `TraceOutputWindow` pools program logic with user interface aspects. For example, the classes related to the traceability graph and the model part for the CCD and text views are tightly coupled with the UI elements in this prototypical implementation.

For three of the four different views within the `TraceOutputWindow` default

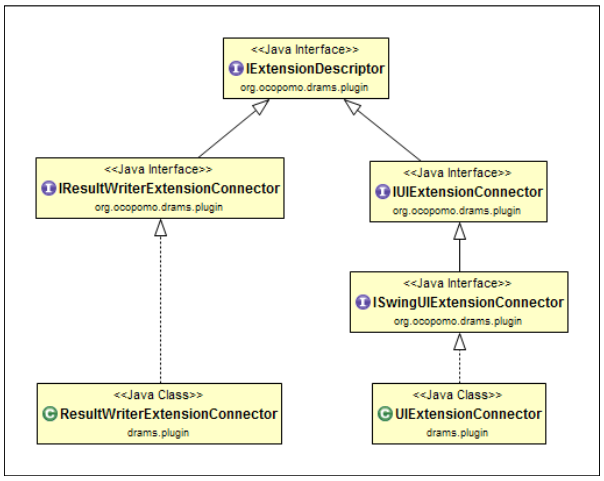

Figure 10.6: Class diagram of plugin extension descriptor

Swing components (or slight modifications of those) are used: A list implementation[116] for the simulation log, a tree view[117] for the CCD and a tabbed pane[118] with text viewers capable of highlighting arbitrary phrases[119] are used.

The main view of the Model Explorer is the visualisation of the traceability graph. For this feature the external framework JGraphX[120] is used. It provides easy, yet quite powerful functionality to display graph data structures, and is therefore a valuable solution for prototyping.

The main program logic of the Model Explorer is based on the graph data model and a graph visiting design pattern. The following subsection sheds some light on the involved classes and algorithms.

Traceability graph and visitor

The main program logic of the Model Explorer is dedicated to transformation of the traceability graph provided by DRAMS into a visualisation graph data structure by a graph visitor implementation. The visualisation graph serves as data model for the JGraphX Visualisation. Purpose of the visitor implementation is — besides creating the visual model — filtering, i.e. to allow visualisation of restricted parts of the traceability graph in order to increase comprehensibility. With such filters e.g. an evaluation tree starting from any rule or fact can be selected and visualised, if desired with a particular range of creation time of facts or time of rule firings.

The visual graph data structure according to Figure 10.8 — as output of the filtering mechanism — is a straightforward object-oriented design. Vertices

[116] Java Swing class `JList`.
[117] Java Swing class `JTree`.
[118] Java Swing class `JTabbedPane`.
[119] `ColorPane`, an extension of the Java Swing class `JTextPane`
[120] `https://github.com/jgraph/jgraphx`

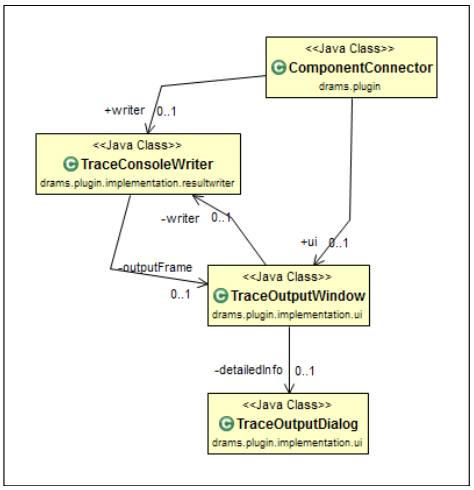

Figure 10.7: Class diagram of user interface components

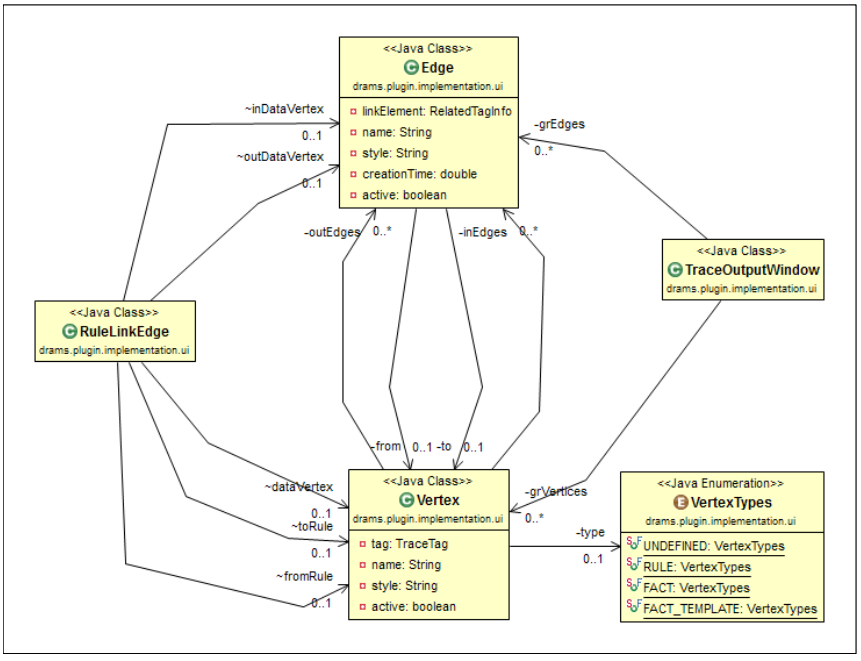

Figure 10.8: Class diagram of visual graph data model

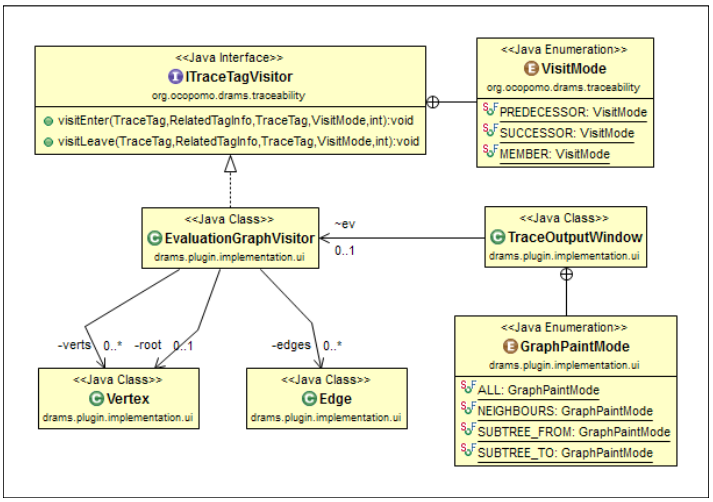

Figure 10.9: Class diagram of trace graph visitor

(class `Vertex`) can either represent rules, facts or fact templates (and can have an 'undefined' state, used as initial state for the transformation process). The `Edge` class realises the links between the vertices.

Within the filtering process, the DRAMS traceability graph is searched by an `EvaluationGraphVisitor` object (Figure 10.9). For each vertex and edge of the traceability graph matching a predefined pattern, respective elements in the visual graph are created. The `EvaluationGraphVisitor` is an implementation of the `ITraceTagVisitor` interface, which defines the standard visitor methods (`visitEnter()` and `visitLeave()`). The interface also defines the visiting mode:

- If the visualisation graph is supposed to contain the evaluation graph for a particular rule firing event or fact, with preceding rule firing events and facts, then the `PREDECESSOR` mode is applied.

- In contrast, for visualising the effects of a particular rule firing event or fact, the `SUCCESSOR` mode results in the generation of a respective deduction graph.

- In order to show related elements (like fact templates for a selected fact), the `MEMBER` mode is used. This mode is reserved for future functionality.

These visitor modes are set by the `TraceOutputWindow` filter mode, defined by `GraphPaintMode`:

- The entire evaluation tree for a particular node is shown when `ALL` is selected.

- With the `NEIGHBOURS` mode, for a selected vertex only the immediate neighbours are displayed, all other graph elements are hidden. This

mode can be enabled in the UI by clicking on a vertex with CTRL and SHIFT keys held down.

- The `SUBTREE_FROM` mode applies the `SUCCESSOR` visitor mode for a selected vertex, by clicking on the vertex with the CTRL key held down.
- The `SUBTREE_TO` model applies the `PREDECESSOR` visitor mode for a selected vertex, by clicking on the vertex with SHIFT key held down.

Time filters are set by dedicated UI elements and applied while generating the visualisation graph. Within this generation process, the visitor not only applies the filter functions, but also gathers information to be displayed in the user interface — for example meta-information is extracted from trace tags, the visualisation styles are determined and the information about the visualised graph elements are routed to the other visualisations — the CCD and text views. These two views highlight the respective parts only for the content of the graph view. An additional feature allows to jump to a highlighted text phrase by double clicking on a CCD annotation. Linking the four different Model Explorer views in the described way enables the user to browse through the traceability information.

10.4 Conclusions

Browsing though the traceability information as described in the last section is basis for exploiting traceability information generated by a simulation model run. This chapter describes the prerequisites and technical solutions for this purpose.

The visualisation of trace information in conjunction with evidence documents — as realised in the DRAMS Model Explorer — makes creating simulation results in form of model-based scenarios easier, and allows to even 'cite' particular phrases, if needed. With this functionality, a significant step towards efficient documentation and transparency of simulation models is done. A more comprehensive picture on this subject is drawn in the following concluding chapter.

It has to be noted here that the DRAMS Model Explorer is a proof-of-concept prototypical implementation. Future work should add more types of visualisations, new filtering mechanisms and other visitor implementations. For example, the static structure of the declarative program could be shown (e.g. as data dependency graph), with (even animated) visualisations of rule activity, unveiling e.g. how often and in which time periods rules or rule clusters fire. Comparing such visualisations for simulation runs with different parameter settings could establish a new dimension of simulation model experimentation and exploration.

Chapter 11

Discussion and conclusions

This chapter concludes this PhD thesis by summarising the results and contributions to the research domains, relating these to the applied methods and providing a critical discussion.

The results from the research as well as the development activities presented in this PhD thesis reflect the different roles the author filled during their creation. Firstly, the role of a software engineer and developer, designing and implementing DRAMS. Secondly, the role of a computational social scientist and 'expert' for agent-based social simulation, designing and realising the use case simulation model.

Most of the work was done in the context of two projects — OCOPOMO and GLODERS — in a number of concrete tasks and yielding tangible results (listed in a chronological order):

- Developing a *simulation tool* as part of the OCOPOMO toolbox, presented in Part III of this PhD thesis.
- Using the OCOPOMO development process and toolbox, mainly from the modeller perspective for *various simulation models*.
- Supporting the *adaptation of the modelling process* for the requirements of GLODERS, mainly from simulation and software architecture viewpoints.
- Developing the *simulation model* as GLODERS deliverable and as basis for Part II of this PhD thesis.
- Supporting experimentation and *result analysis* with this model, in particular concerning *traceability*.

All these activities were performed rigorously applying the associated methods of computational social science and software engineering. For the latter two activities, simultaneously performing in the two roles defined above was considerably beneficial. This leads to the main goal of this PhD thesis, to integrate the perspectives of the different roles into an — as far as possible —

coherent picture. As such, it can be seen as a contribution to applied computer science and software engineering, in particular the technical realisation of traceability within a development process, as presented in Chapter 10.

The concrete results of the PhD thesis can be elicited with view on the original research questions. Four general overarching research questions for this PhD thesis are defined in Chapter 1, while more specific research questions for the use case simulation model are noted down in Chapter 5. For the latter, research questions are formulated from stakeholder, social scientist and computer scientist perspectives, and partly answered already in Chapter 7 for the stakeholder and social science questions. What remains to be done is the discussion of the research questions related to computer science.

Firstly, regarding the general research questions the following can be stated:

RQ 1: How must a modelling process for developing evidence-based social simulation models be designed, taking questions about stakeholder involvement, requirement elicitation, model implementation, verification and validation into account?

Results and discussion: Core element of such a process must be some kind of conceptual model, which allows to specify the model target in a way formal enough to be useful to inform program code for the model, and at the same time informative enough to be used as a means to communicate model aspects to involved stakeholders. The process must include a phase for the development of the conceptual model, which is dependent on the model grounds, e.g. theory or evidence. To provide input for this phase, an evidence collection and preprocessing phase might be needed for a structured approach to evidence management. Furthermore, different (manual or automated) approaches to transforming the conceptual model into verified formal simulation model code must be supported in another phase. A final phase is needed to generate 'productive' simulation results to be validated.

The process must support to go back and forth between the different phases, and in case the evidence contains variable facts or views (as in future scenarios for policies), it must support to start the modelling cycle again with updated evidences, for example if validation 'failed'.

These requirements are fulfilled with the OCOPOMO model development process as presented in Chapter 3. The design of this process has proven to be applicable and useful for several different pilot and use case models, also in other areas than policy modelling, as shown in Chapter 4.

RQ 2: What are the appropriate methods to be applied within the different modelling process phases that are aligned with the questions referred to in RQ 1?

Results and discussion: For the conceptual modelling phase, basically all according methods can be applied, for example ontologies, UML diagrams, or simulation-specific solutions such as the CCD. The methods applied to collect and process information to be included in the conceptual model depend on the kind of data and the envisaged application. The scenario method has proven to be useful for policy models (see section 3.5), while other model types require qualitative or quantitative data analysis methods (see section 4.5). The transformation into formal model code can be supported by intrinsic features of the conceptual model (such as in Executable UML), or rely on code transformation — preferably based upon a model-to-model transformation scheme, but the 'less expensive' model-to-code approach is also useful. Related to the transformation method, an appropriate formalisation approach must be selected, that supports the envisaged model features. Agent-based simulation is one example, others might include macroscopic or multi-level simulation approaches.

The most important method in this respect is an encompassing traceability, preserving links between the artefacts of the different phases.

Such a selection of methods has been applied for the successful use case model development presented in Part II and concluded in section 7.6.

RQ 3: How can the methods according to RQ 2 be supported by software tools?

Results and discussion: This question is partly answered by the practical results of the work presented in this PhD thesis. For the conceptual modelling phase, the CCD tool (see section 3.3.2) fulfils the specific requirements very well and provides added value for all involved parties:

Firstly, the modellers get an instrument at hand to structure information and knowledge. In addition, they can count on a decreased workload due to automated generation of significant ('boring' structural) parts of model code, so they can directly start with implementing the 'interesting' rule content.

Secondly, the stakeholders can dive much deeper into model details than it would be possible from narrative documentation, if they don't want to read program code. Stakeholders in particular with research background might want to follow the modelling and simulation process beyond the evidence provision and simulation model validation. In such cases the communication aspect of the conceptual model is crucial. This unfortunately implies that the contents have to be less formal, which impairs automated code transformation. Here, more research is needed to overcome this

gap. For example, another conceptual layer could be introduced, providing common concepts of formalisations, that can serve as links between informal concepts and code building blocks of various implementation styles and programming languages.

Also for the model implementation phase, this PhD thesis provides an answer with DRAMS, presented in Part III. Defining agent behaviour as declarative rules is well aligned with the concepts provided by the CCD. Still an open question is how the dependencies between conceptual model and simulation model code can be made independent from concrete simulation approaches (such as agent-based simulation in this particular case).

Tools for methods in context to evidence collection and processing are also dependent on the particular method. In this area (of e.g. data analysis), typically many options to choose between are available. Main catch in this respect are the often restricted means for integrating such tools smoothly into the tool chain, e.g. by providing interfaces to be used by the conceptual modelling tool.

Most importantly, appropriate tool support must ensure the technical realisation of traceability (with an example presented in Chapter 10).

RQ 4: How can process (according to RQ 1), methods (according to RQ 2) and tools (according to RQ 3) be applied for creating a specific simulation model in the domain of criminology?

Results and discussion: As a short answer, by using the right methods and tools while applying the OCOPOMO process, as discussed in Chapter 4. It is important in this context to tailor the evidence generation to the kind of evidence documents available (which can appear in a possibly wide range, from unstructured wire-tapping or interrogation protocols, to structured communication meta-data). As in many domains with restricted affinity to computer science (such as in management, politics), it is important to face police stakeholders with information on the modelling and simulation procedures they can comprehend. Showing the program code will never pay off in such cases; hence, the purpose of the conceptual model as a communication means becomes particularly important — in a way to provide the basis of a common terminology, an ontology.

Another crucial aspect is the adequate presentation of the simulation results, as reasoned in Chapter 7. Not only that language the police stakeholders are familiar with becomes even more important. As a conclusion it can be said that content, structure and language of the presented results should always be similar to the appearance of the evidence documents. Following this approach makes it much more likely that the results are actually carefully

examined, discussed and scrutinised, so that the model-based scenarios created with the simulations were even regarded by police officers as 'virtual experience', paving the way to opening minds for considering new views on seemingly already 'concluded knowledge'.

Having these answers in mind, the following can be said regarding the unanswered research questions for the simulation model, complementing the discussion in section 7.6:

Use Case CS RQ 1: Is it possible to apply a stringent modelling process inspired from software engineering for the realisation of complex simulation models which can be regarded as a special class of software systems with typically soft and blur requirements, while these requirements are not only biased through stakeholder engagement (which is the baseline e.g. for industrial applications), but also during the process of conceptual modelling, implementation and execution of simulations, triggered by verification and validation of the developed software, and in addition by theories of human behaviour?

Results and discussion: This question is a summary of the general research questions discussed above, concretised with assumptions regarding the peculiarities of simulation models viewed as software systems. This is related to discussions about the restricted usefulness of extensive software specification in cases where the system to develop is a moving target, because the requirements are changing rapidly — in the case of a simulation model, even by program execution itself.

A fully satisfying solution for this special case has not been found yet and needs further research.

Use Case CS RQ 2: Does thorough and comprehensive software specification result in added value regarding quality of both model implementation and simulation results?

Results and discussion: The answer to this question needs to be differentiated. On the one hand, this approach was successfully followed developing the use case model, with the specification as a valuable means for discussions among involved persons, resulting in a well-documented and comprehensive model implementation and highly relevant model results (as confirmed by the police stakeholders). On the other hand, the problem with rapidly changing requirements and the not fully closed tool chain put some burden on the model developer, as respective changes had to be kept consistent in all the respective artefacts.

Use Case CS RQ 3: Does traceability from simulation results back to evidence documents increase the acceptance of the simulation method for stakeholders?

Results and discussion: This question deserves a clear Yes. The model-based scenarios could only be written in this style (riddled with actual text fragments from evidence documents) using traceability features.

Concluding it can be said that the research questions led to interesting and comprehensible results and to significant contributions not only to the domains of criminology and computational social science (and, thus, also for applied computer science), but also for software engineering as a discipline of computer science in general: The example how to connect traceability with a model development process and associated tools also applies more or less unaltered to general evidence-driven software development processes, involving structured methods to elicit user requirements for software systems.

Appendix

Appendix A

Verification and validation techniques

Informal	**Static**	**Dynamic**	**Formal**
Audit	Cause-effect graphing	Acceptance testing	Induction
Desk checking	Control analysis	Alpha testing	Inductive assertions
Documentation checking	Data analysis	Assertion checking	Inference
Face validation	Fault/failure analysis	Beta testing	Logical deduction
Inspections	Interface analysis	Bottom-up testing	Lambda calculus
Reviews	Semantic analysis	Comparison testing	Predicate calculus
Turing test	Structural analysis	Statistical techniques	Predicate transformation
Walkthroughs	Symbolic analysis	Structural testing	Proof of correctness
	Syntax analysis	Submodel / module testing	
	Traceability assessment	Visualization / animation	

Table A.1: Verification and validation techniques, copied from (Sokolowski and Banks, 2010, p. 16)

Appendix B

Use case model supplements

The use case simulation model and some additional documentation can be obtained from `http://www.lotzmann.net/PhD/AppendixB/ModelCodeRepo`.

Table B.1 shows structure and content of the model code repository.

Folder or file	**Content**
data	folder to store simulation outcomes for simulation runs
documentation	additional model documentation
documents	evidence documents
DRAMS	DRAMS runtime package with plugins and external libraries
simulation-src	simulation model source code
*_v1.ccd	main CCD file for simulation model
*_v1.ccd_actions	CCD action diagram of simulation model
*_v1.ccd_diagram	CCD actor-network diagram of simulation model

Table B.1: Structure an content of the use case model repository

Appendix C

DRAMS supplements

C.1 Language specification and guide

The DRAMS manual (an updated version of (Lotzmann and Meyer, 2013)) can be downloaded from `http://www.lotzmann.net/PhD/AppendixC/DRAMSManual`.

C.2 Code repository

The DRAMS code repository can be obtained from `http://www.lotzmann.net/PhD/AppendixC/DRAMSCodeRepo`.

Table C.1 shows structure and content of the repository.

C.3 Test model

The test model can be obtained from `http://www.lotzmann.net/PhD/AppendixC/DRAMSTestModel`.

Folder	**Content**
documentation	DRAMS user manual and other documentation
DRAMS	DRAMS source code
— DRAMS-developer	Eclipse project containing the DRAMS sources DRAMS
— DRAMS-platforms	Platform-specific model class (for RepastJ)
— DRAMS-testing	DRAMS test model
plugins	DRAMS plugin source code
— ConsoleWriterPlugin	UI plugin providing a simple log window
— CSVWriterPlugin	Output writer plugin for CSV generation
— DomUL	custom DOM implementation
— DRAMSConsolePlugin	DRAMS Console
— DRAMSModelExplorer	DRAMS Model Explorer
— DRAMSSwingGUIPlugin	DRAMS UI
— MultiTabConsoleWriterPlugin	UI plugin providing log window with different views
— NumericalXMLWriterPlugin	Output writer plugin for XML generation
— SwingGUIComponents	commong GUI classes
— TextWriterPlugin	Output writer plugin for text generation

Table C.1: Structure an content of the DRAMS repository

Bibliography

Abar, S., Theodoropoulos, G. K., Lemarinier, P., and O'Hare, G. M. (2017). Agent based modelling and simulation tools: A review of the state-of-art software. *Computer Science Review*, 24:13–33.

Aggarwal, C. C. (2015). *Data Mining: The Textbook*. Springer Publishing Company, Incorporated.

Aggarwal, C. C. and Zhai, C., editors (2012). *Mining Text Data*. Springer.

Aho, A. V., Lam, M. S., Sethi, R., and Ullman, J. D. (2006). *Compilers: Principles, Techniques, and Tools (2nd Edition)*. Addison-Wesley, Boston, MA, USA.

Alam, S. J., Meyer, R., Ziervogel, G., and Moss, S. (2007). The impact of HIV/AIDS in the context of socioeconomic stressors: an evidence-driven approach. *Journal of Artificial Societies and Social Simulation*, 10(4):7. `http://jasss.soc.surrey.ac.uk/10/4/7.html`.

Alvertis, I., Koussouris, S., Papaspyros, D., Arvanitakis, E., Mouzakitis, S., Franken, S., Kolvenbach, S., and Prinz, W. (2016). User involvement in software development processes. *Procedia Computer Science*, 97:73–83. 2nd International Conference on Cloud Forward: From Distributed to Complete Computing.

Alves Furtado, B. and Eberhardt, I. D. R. (2016). A simple agent-based spatial model of the economy: Tools for policy. *Journal of Artificial Societies and Social Simulation*, 19(4):12. `http://jasss.soc.surrey.ac.uk/19/4/12.html`.

Andrighetto, G., Brandts, J., Conte, R., Sabater-Mir, J., Solaz, H., and Villatoro, D. (2013). Punish and voice: Punishment enhances cooperation when combined with norm-signalling. *Plos one*, 8(6):e64941.

Andrighetto, G., Nardin, L. G., Lotzmann, U., and Neumann, M. (2014). D 3.1 report on adaptations made to the emil simulator. Deliverable 3.1. FP7 project GLODERS (Global Dynamics of Extortion Racket Systems). Deliverable. `http://www.gloders.eu/images/Deliverables/GLODERS_D3-1.pdf` (last accessed 24/2/2019).

Antinori, A. (2008). Information communication technology & crime: the future of criminology. *Rivista di Criminologia, Vittimologia e Sicurezza*, 2(3):23–31.

Anzola, D., Neumann, M., Möhring, M., and Troitzsch, K. G. (2016). National mafia-type organisations: Local threat, global reach. In Elsenbroich, C., Anzola, D., and Gilbert, N., editors, *Social Dimensions of Organised Crime: Modelling the Dynamics of Extortion Rackets*, pages 9–23. Springer International Publishing, Cham.

Avison, D. E., Golder, P. A., and Shah, H. U. (1992). Towards an SSM toolkit: rich picture diagramming. *European Journal of Information Systems*, 1(6):397–408.

Axelrod, R. (1997). The dissemination of culture: A model with local convergence and global polarization. *Journal of Conflict Resolution*, 41(2):203–226.

Axelrod, R. (2003). Advancing the Art of Simulation in the Social Sciences. *Japanese Journal for Management Information Systems*, 12(2):1–19.

Banks, J. (1998). *Handbook of Simulation*. Wiley, New York, NY.

Banks, J., Carson, J. S., Nelson, B. L., and Nicol, D. M. (2009). *Discrete-Event System Simulation (5th Edition)*. Prentice Hall, 5 edition.

Baptista, M., Roque Martinho, C., Lima, F., A. Santos, P., and Prendinger, H. (2014). Improving learning in business simulations with an agent-based approach. *Journal of Artificial Societies and Social Simulation*, 17(3):7. `http://jasss.soc.surrey.ac.uk/17/3/7.html`.

Barreteau, O. (2003). Our companion modelling approach. *Journal of Artificial Societies and Social Simulation*, 6(2):1. `http://jasss.soc.surrey.ac.uk/6/2/1.html`.

Barthelemy, J. and Toint, P. (2015). A stochastic and flexible activity based model for large population. Application to Belgium. *Journal of Artificial Societies and Social Simulation*, 18(3):15. `http://jasss.soc.surrey.ac.uk/18/3/15.html`.

Barthelemy, O., Moss, S., Downing, T., and Rouchier, J. (2001). Policy modelling with ABSS: The case of water demand management. CPM Report 02-92, Centre for Policy Modelling, Manchester Metropolitan University, Manchester.

Barton, R. R. (2010). Simulation experiment design. In *Proceedings of the Winter Simulation Conference*, WSC '10, pages 75–86. Winter Simulation Conference.

Beck, K. (1999). Embracing change with extreme programming. *Computer*, 32(10):70–77.

Beck, K. and Andres, C. (2004). *Extreme Programming Explained: Embrace Change (2nd Edition)*. Addison-Wesley Professional.

Bednár, P. and Hartenfels, C. (2013). D4.2 system and user documentation. SD-1: System documentation of ocopomo alfresco tools. Deliverable 4.2, Annex SD-1 v1.0, OCOPOMO consortium. `http://www.ocopomo.eu/results/public-deliverables/system-documentation/d4.2-sd-1-system-documentation-of-ocopomo-alfresco-tools.pdf` (last accessed 24/2/2019).

Beltran, F. S., Herrando, S., Ferreres, D., Adell, M.-A., Estreder, V., and Ruiz-Soler, M. (2009). Forecasting a language shift based on cellular automata. *Journal of Artificial Societies and Social Simulation*, 12(3):5. `http://jasss.soc.surrey.ac.uk/12/3/5.html`.

Bennett, A. (2000). *Popular Music and Youth Culture: Music, Identity and Place*. Palgrave Macmillan.

Berners-Lee, T., Fielding, R., and Masinter, L. (2005). RFC 3986, Uniform Resource Identifier (URI): Generic Syntax. `https://www.ietf.org/rfc/rfc3986.txt` (last accessed 24/2/2019).

Bicking, M., Butka, P., Delrio, C., Dunilova, V., Hilovska, K., Kacprzyk, M., Lotzmann, U., Łucznik, K., Mach, M., Moss, S., Nowak, A., Pizzo, C., Rinaldi, V., Roszczynska-Kurasinska, M., Sabol, T., Scherer, S., Schmidt, A., Ventzke, S., and Wimmer, M. A. (2010). D1.1 stakeholder identification and requirements for toolbox, scenario process and policy modelling. Deliverable 1.1 v1.0, OCOPOMO consortium. `http://www.ocopomo.eu/results/public-deliverables/OCOPOMO_D1.1_v10.pdf` (last accessed 24/2/2019).

Bicking, M., Delrio, C., Dulinova, V., Kacprzyk, M., Lotzmann, U., Moss, S., Neuroth, C., Pinotti, D., Roszczynska, M., Scherer, S., Tapak, P., and Wimmer, M. A. (2013). D6.1 narrative scenarios and policy models. Deliverable 6.1 v1.2, OCOPOMO consortium. `http://www.ocopomo.eu/results/public-deliverables/d-6.1-narrative-scenarios-and-policy-models` (last accessed 24/2/2019).

Boehm, B. W. (1979). Guidelines for verifying and validating software requirements and design specifications. In Samet, P. A., editor, *Euro IFIP 79*, pages 711–719. North Holland.

Bommel, P., Dieguez, F., Bartaburu, D., Duarte, E., Montes, E., Pereira Machín, M., Corral, J., Lucena, C. J. P. d., and Morales Grosskopf, H. (2014). A further step towards participatory modelling. Fostering

stakeholder involvement in designing models by using executable UML. *Journal of Artificial Societies and Social Simulation*, 17(1):6. `http://jasss.soc.surrey.ac.uk/17/1/6.html`.

Booch, G., Rumbaugh, J., and Jacobson, I. (2005). *The unified modeling language user guide*. Addison-Wesley, Upper Saddle River, NJ.

Bosse, T. and Gerritsen, C. (2008). Agent-based simulation of the spatial dynamics of crime: On the interplay between criminal hot spots and reputation. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '08, pages 1129–1136, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

Bourgais, M., Taillandier, P., Vercouter, L., and Adam, C. (2018). Emotion modeling in social simulation: A survey. *Journal of Artificial Societies and Social Simulation*, 21(2):5. `http://jasss.soc.surrey.ac.uk/21/2/5.html`.

Braun, N. and Saam, N. (2014). *Handbuch Modellbildung und Simulation in den Sozialwissenschaften*. Springer Fachmedien Wiesbaden.

Brenner, T. and Werker, C. (2009). Policy advice derived from simulation models. *Journal of Artificial Societies and Social Simulation*, 12(4):2. `http://jasss.soc.surrey.ac.uk/12/4/2.html`.

Bright, D. A., Hughes, C. E., and Chalmers, J. (2012). Illuminating dark networks: a social network analysis of an Australian drug trafficking syndicate. *Crime, Law and Social Change*, 57(2):151–176.

Bryman, A. (2006). Integrating quantitative and qualitative research: how is it done? *Qualitative Research*, 6(1):97–113.

Buckheit, J. B. and Donoho, D. L. (1995). WaveLab and reproducible research. In Antoniadis, A. and Oppenheim, G., editors, *Wavelets and Statistics*, pages 55–81. Springer New York, New York, NY.

Burg, M. B., Peeters, H., and Lovis, W. A. (2016). *Uncertainty and Sensitivity Analysis in Archaeological Computational Modeling*. Springer Publishing Company, Incorporated, 1st edition.

Butka, P., Mach, M., Furdik, K., and Genci, J. (2011). Design of a System Architecture for Support of Collaborative Policy Modelling Processes. In *Proceedings of SACI 2011, 6th IEEE International Symposium on Applied Computational Intelligence and Informatics*, pages 193–198, Red Hook, NY, USA. Óbuda University, Hungary, IEEE, Curran Associates, Inc.

Byrne, J. and Marx, G. (2011). Technological innovations in crime prevention and policing. A review of the research on implementation and impact. *Journal of Police Studies*, 3(20):17–40.

Camus, B., Bourjot, C., and Chevrier, V. (2015). Considering a multi-level model as a society of interacting models: Application to a collective motion example. *Journal of Artificial Societies and Social Simulation*, 18(3):7. `http://jasss.soc.surrey.ac.uk/18/3/7.html`.

Carroll, J. (1995). *Scenario-based design: envisioning work and technology in system development*. Wiley.

Castelfranchi, C. (1995). Guarantees for autonomy in cognitive agent architecture. In *Proceedings of the Workshop on Agent Theories, Architectures, and Languages on Intelligent Agents*, ECAI-94, pages 56–70, Berlin, Heidelberg. Springer-Verlag.

Cetinkaya, D. (2013). *Model Driven Development of Simulation Models - Defining and Transforming Conceptual Models into Simulation Models by Using Metamodels and Model Transformations*. PhD thesis, Technische Universiteit Delft.

Cetinkaya, D. and Verbraeck, A. (2011). Metamodeling and model transformations in modeling and simulation. In *Proceedings of the Winter Simulation Conference*, WSC '11, pages 3048–3058. Winter Simulation Conference.

Cetinkaya, D., Verbraeck, A., and Seck, M. D. (2011). MDD4MS: A model driven development framework for modeling and simulation. In *Proceedings of the 2011 Summer Computer Simulation Conference*, SCSC '11, pages 113–121, Vista, CA. Society for Modeling & Simulation International.

Chattoe, E., Saam, N. J., and Möhring, M. (2000). Sensitivity analysis in the social sciences: Problems and prospects. In Suleiman, R., Troitzsch, K. G., and Gilbert, N., editors, *Tools and techniques for social science simulation*, pages 243–273, Heidelberg. Physica-Verlag.

Cioffi-Revilla, C. (2010). Computational social science. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(3):259–271.

Cleland-Huang, J. and Mobasher, B. (2008). Using data mining and recommender systems to scale up the requirements process. In *Proceedings of the 2Nd International Workshop on Ultra-large-scale Software-intensive Systems*, ULSSIS '08, pages 3–6, New York, NY, USA. ACM.

Codagnone, C. and Wimmer, M. A., editors (2007). *Roadmapping eGovernment Research: Visions and Measures towards Innovative Governments in 2020*. MY Print snc di Guerinoni Marco & C.

Coen, C. A. (2009). Simple but not simpler. Introduction CMOT special issue - simple or realistic. *Computational and Mathematical Organization Theory*, 15(1):1–4.

Colquitt, J. A., Lepine, J. A., Piccolo, R. F., Zapata, C. P., and Rich, B. L. (2012). Explaining the justice-performance relationship: trust as exchange deepener or trust as uncertainty reducer? *The Journal of applied psychology*, 97 1:1–15.

Conte, R., Andrighetto, G., and Campennì, M. (2013). *Minding Norms: Mechanisms and Dynamics of Social Order in Agent Societies*. Oxford Series on Cognitive Models and Architectures. OUP USA.

Conte, R. and Paolucci, M. (2001). Intelligent social learning. *Journal of Artificial Societies and Social Simulation*, 4(1):3. `http://jasss.soc.surrey.ac.uk/4/1/3.html`.

Corbin, J. M. and Strauss, A. L. (2008). *Basics of qualitative research*. Sage Publ., 3 edition.

Crooks, A., Hudson-Smith, A., and Dearden, J. (2009). Agent street: An environment for exploring agent-based models in second life. *Journal of Artificial Societies and Social Simulation*, 12(4):10. `http://jasss.soc.surrey.ac.uk/12/4/10.html`.

Dean, J., Gumerman, G., Epstein, J., Axtell, R., Swedlund, A., Parker, M., and Mccarroll, S. (2012). Understanding anasazi culture change through agent-based modeling. In *Generative Social Science: Studies in Agent-Based Computational Modeling*, pages 90–116. Princeton University Press.

Delcambre, L. M. L., Liddle, S. W., Pastor, O., and Storey, V. C. (2018). A reference framework for conceptual modeling. In *Conceptual Modeling - 37th International Conference, ER 2018, Xi'an, China, October 22-25, 2018, Proceedings*, pages 27–42.

Diesner, J. and Carley, K. M. (2005). Revealing social structure from texts: Meta-matrix text analysis as a novel method for network text analysis. In Narayanan, V. and Armstrong, D., editors, *Causal Mapping for Information Systems and Technology Research: Approaches, Advances, and Illustrations*, pages 81–108. Idea Group Publishing.

Dignum, V. and Dignum, F. (2013). *Perspectives on Culture and Agent-based Simulations: Integrating Cultures*. Studies in the Philosophy of Sociality. Springer International Publishing.

Dijkstra, P., Bex, F., Prakken, H., and de Vey Mestdagh, K. (2005). Towards a multi-agent system for regulated information exchange in crime investigations. *Artificial Intelligence and Law*, 13(1):133–151.

Dilaver, Ö. (2015). From participants to agents: Grounded simulation as a mixed-method research design. *Journal of Artificial Societies and Social Simulation*, 18(1):15. `http://jasss.soc.surrey.ac.uk/18/1/15.html`.

Drogoul, A., Vanbergue, D., and Meurisse, T. (2003). Simulation orientée agent: où sont les agents? Technical report, LIP6 Université Paris 6.

Edmonds, B. (2015a). A context- and scope-sensitive analysis of narrative data to aid the specification of agent behaviour. *Journal of Artificial Societies and Social Simulation*, 18(1):17. `http://jasss.soc.surrey.ac.uk/18/1/17.html`.

Edmonds, B. (2015b). Using qualitative evidence to inform the specification of agent-based models. *Journal of Artificial Societies and Social Simulation*, 18(1):18. `http://jasss.soc.surrey.ac.uk/18/1/18.html`.

Edmonds, B. and Moss, S. (2004). From KISS to KIDS - an 'anti-simplistic' modelling approach. In Davidsson, P., Logan, B., and Takadama, K., editors, *MABS*, volume 3415 of *Lecture Notes in Computer Science*, pages 130–144. Springer.

Edmonds, B. and Wallis, S. (2002). Towards an ideal social simulation language. Technical report, Manchester Metropolitan University.

Elsawi, A. M., Shahibudin, S., and Ibrahim, R. (2015). Model driven architecture a review of current literature. *Journal Of Theoretical And Applied Information Technology*, 79(1):122–127.

Elsenbroich, C. (2017). The addio pizzo movement: exploring social change using agent-based modelling. *Trends in Organized Crime*, 20(1):120–138.

Elsenbroich, C., Anzola, D., and Gilbert, N. (2016). Introduction. In Elsenbroich, C., Anzola, D., and Gilbert, N., editors, *Social Dimensions of Organised Crime: Modelling the Dynamics of Extortion Rackets*, pages 1–6. Springer International Publishing, Cham.

Embley, D. W., Liddle, S. W., and Pastor, O. (2011). Conceptual-model programming: A manifesto. In Embley and Thalheim (2011), pages 3–16.

Embley, D. W. and Thalheim, B., editors (2011). *Handbook of Conceptual Modeling - Theory, Practice, and Research Challenges*. Springer.

Epstein, J. M. (2007). *Generative Social Science: Studies in Agent-Based Computational Modeling (Princeton Studies in Complexity)*. Princeton University Press, Princeton, NJ, USA.

Epstein, J. M. and Axtell, R. L. (1996). *Growing Artificial Societies: Social Science from the Bottom Up*. Complex adaptive systems. The MIT Press, Cambridge, MA.

Evertsz, R., Thangarajah, J., and Papasimeon, M. (2017). The conceptual modelling of dynamic teams for autonomous systems. In Mayr, H. C., Guizzardi, G., Ma, H., and Pastor, O., editors, *Conceptual Modeling*, pages 311–324, Cham. Springer International Publishing.

Fieldhouse, E., Lessard-Phillips, L., and Edmonds, B. (2016). Cascade or echo chamber? A complex agent-based simulation of voter turnout. *Party Politics*, 22(2):241–256.

Filho, C. S. d. F. and Ramalho, G. (2000). JEOPS - the java embedded object production system. In *Proceedings of the International Joint Conference, 7th Ibero-American Conference on AI: Advances in Artificial Intelligence*, IBERAMIA-SBIA '00, pages 53–62, London, UK. Springer-Verlag.

Fischer, K. and Warwas, S. (2013). A methodological approach to model driven design of multiagent systems. In Müller, J. P. and Cossentino, M., editors, *Agent-Oriented Software Engineering XIII*, pages 1–21, Berlin, Heidelberg. Springer Berlin Heidelberg.

Fishwick, P. A. (1995). *Simulation Model Design and Execution: Building Digital Worlds*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition.

Forgy, C. L. (1981). OPS5 user's manual. Technical report, Carnegie-Mellon Univ Pittsburgh PA Dept of Computer Science. `http://www.dtic.mil/dtic/tr/fulltext/u2/a106558.pdf` (last accessed 24/2/2019).

Forgy, C. L. (1982). Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19(1):17–37.

Friedman-Hill, E. (2003). *Jess in Action: Rule-Based Systems in Java*. Manning Publications Co.

Furdik, K., Smatana, P., Bednar, P., Butka, P., Lotzmann, U., Scherer, S., Furdikova, V., Smatana, J., and Hartenfels, C. (2013). D4.2 system and user documentation. Deliverable 4.2 v1.0, OCOPOMO consortium. `http://www.ocopomo.eu/results/public-deliverables/d4.2-system-and-user-documentation` (last accessed 24/2/2019).

Galvis Carreño, L. V. and Winbladh, K. (2013). Analysis of user comments: An approach for software requirements evolution. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 582–591, Piscataway, NJ, USA. IEEE Press.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Geertz, C. (1973). *Thick description: Toward an interpretive theory of culture*, pages 3–30. Basic Books, New York.

Genesereth, M. R. and Ketchpel, S. P. (1994). Software agents. *Commun. ACM*, 37(7):48–ff.

Gerritsen, C. (2015). Agent-based modelling as a research tool for criminological research. *Crime Science*, 4(1):2.

Ghorbani, A., Dijkema, G., and Schrauwen, N. (2015). Structuring qualitative data for agent-based modelling. *Journal of Artificial Societies and Social Simulation*, 18(1):2. `http://jasss.soc.surrey.ac.uk/18/1/2.html`.

Gibbs, J. P. (1965). Norms: The problem of definition and classification. *American Journal of Sociology*, 70(5):586–594.

Gilbert, N. (2007). Computational social science: Agent-based social simulation. In Phan, D. and Amblard, F., editors, *Agent-based Modelling and Simulation*, pages 115–134. Bardwell, Oxford.

Gilbert, N., Ahrweiler, P., Barbrook-Johnson, P., Narasimhan, K. P., and Wilkinson, H. (2018). Computational modelling of public policy: Reflections on practice. *Journal of Artificial Societies and Social Simulation*, 21(1):14. `http://jasss.soc.surrey.ac.uk/21/1/14.html`.

Gilbert, N. and Troitzsch, K. G. (2005). *Simulation for the Social Scientist. 2nd edition.* Open University Press, McGraw-Hill, Maidenhead.

Given, L. (2008). *The SAGE Encyclopedia of Qualitative Research Methods.* Thousand Oaks, California.

Glaser, B. G. and Strauss, A. L. (1967). *The Discovery of Grounded Theory: Strategies for Qualitative Research.* Aldine de Gruyter, New York, NY.

Gotel, O. C. Z. and Finkelstein, C. W. (1994). An analysis of the requirements traceability problem. In *Proceedings of IEEE International Conference on Requirements Engineering*, pages 94–101.

Greve, J., Schnabel, A., and Schützeichel, R. (2009). *Das Mikro-Makro-Modell der soziologischen Erklärung: Zur Ontologie, Methodologie und Metatheorie eines Forschungsprogramms.* VS Verlag für Sozialwissenschaften.

Grimm, V. and Railsback, S. F. (2005). *Individual-based modeling and ecology.* Princeton University Press, Princeton.

Groff, E. R., Johnson, S. D., and Thornton, A. (2018). State of the art in agent-based modeling of urban crime: An overview. *Journal of Quantitative Criminology.*

Gruber, T. (2009). Ontology. In Liu, L. and Özsu, M. T., editors, *Encyclopedia of Database Systems*, pages 1963–1965. Springer US.

Hannappel, M. (2015). *(K)ein Ende der Bildungsexpansion in Sicht!? Ein Mikrosimulationsmodell zur Analyse von Wechselwirkungen zwischen demographischen Entwicklungen und Bildungsbeteiligung*. Metropolis-Verlag, Marburg.

Hayes, J. H., Dekhtyar, A., and Sundaram, S. (2005). Text mining for software engineering: How analyst feedback impacts final results. *SIGSOFT Softw. Eng. Notes*, 30(4):1–5.

Helbing, D. and Johansson, A. (2007). Quantitative agent-based modeling of human interactions in space and time. In Amblard, F., editor, *Fourth Conference of The European Social Simulation Association (ESSA)*, pages 623–637. Institut de Recherche en Informatique de Toulouse (IRIT), Toulouse.

Hilbert, M. (2015). e-Science for Digital Development:"ICT4ICT4D": Development Informatics Working Paper no.60.

Hummon, N. P. and Fararo, T. J. (1995). The emergence of computational sociology. *The Journal of mathematical sociology*, 20(2-3):79–87.

IEEE (1990). IEEE standard glossary of software engineering terminology. *IEEE Std 610.12-1990*, pages 1–84.

Interis, M. (2011). On norms: A typology with discussion. *American Journal of Economics and Sociology*, 70(2):424–438.

Isabella, L. A. (1990). Evolving interpretations as a change unfolds: How managers construe key organizational events. *Academy of Management Journal*, 33(1):7.

Jackson, P. J. (2014). Software solutions for computational modelling in the social sciences. In Dabbaghian, V. and Mago, V. K., editors, *Theories and Simulations of Complex Social Systems*, volume 52 of *Intelligent Systems Reference Library*, pages 5–17. Springer.

Jones, N. A., Perez, P., Measham, T. G., Kelly, G. J., d'Aquino, P., Daniell, K. A., Dray, A., and Ferrand, N. (2009). Evaluating participatory modeling: Developing a framework for cross-case analysis. *Environmental Management*, 44(6):1180.

Kallel, S., Loulou, M., Rekik, M., and Kacem, A. H. (2013). MDA-based approach for implementing secure mobile agent systems. In Müller, J. P. and Cossentino, M., editors, *Agent-Oriented Software Engineering XIII*, pages 56–72, Berlin, Heidelberg. Springer Berlin Heidelberg.

Kang, H.-W. and Kang, H.-B. (2017). Prediction of crime occurrence from multi-modal data using deep learning. In *PloS one*.

Kelly, D. F. (2007). A software chasm: Software engineering and scientific computing. *IEEE Software*, 24(6):120–119.

Kheir, N. A. (1988). *Systems Modeling and Computer Simulation*. Marcel Dekker, Inc., New York, NY, USA.

Klügl, F. (2010). *Agent-Based Simulation Engineering*. Habilitation thesis, University of Würzburg, Germany.

Klügl, F. (2013). "Engineering" Agent-Based Simulation Models? In Müller, J. P. and Cossentino, M., editors, *Agent-Oriented Software Engineering XIII*, pages 179–196, Berlin, Heidelberg. Springer Berlin Heidelberg.

Kohler, T. A. and Gummerman, G. J., editors (2001). *Dynamics of Human and Primate Societies: Agent-Based Modeling of Social and Spatial Processes*. Oxford University Press, Inc., New York, NY, USA.

Kowalska-Styczeń, A. and Sznajd-Weron, K. (2016). From consumer decision to market share — unanimity of majority? *Journal of Artificial Societies and Social Simulation*, 19(4):10. `http://jasss.soc.surrey.ac.uk/19/4/10.html`.

Kowalski, R. A. (1988). The early years of logic programming. *Commun. ACM*, 31(1):38–43.

Krukow, O., Neumann, M., Lotzmann, U., and Möhring, M. (2014). D 2.1 analysis of content and structure of data leading to a shared ontology. Deliverable 2.1. FP7 project GLODERS (Global Dynamics of Extortion Racket Systems). Deliverable. `http://www.gloders.eu/images/Deliverables/GLODERS_D2-1.pdf` (last accessed 24/2/2019).

Latour, B. (2005). *Reassembling the Social:An Introduction to Actor-Network-Theory: An Introduction to Actor-Network-Theory*. Clarendon Lectures in Management Studies. OUP Oxford.

Le Page, C., Bobo, K. S., Kamgaing, T. O. W., Ngahane, B. F., and Waltert, M. (2015). Interactive simulations with a stylized scale model to codesign with villagers an agent-based model of bushmeat hunting in the periphery of Korup National Park (Cameroon). *Journal of Artificial Societies and Social Simulation*, 18(1):8. `http://jasss.soc.surrey.ac.uk/18/1/8.html`.

Leach, P., Mealling, M., and Salz, R. (2005). RFC 4122: A Universally Unique IDentifier (UUID) URN Namespace. `http://www.ietf.org/rfc/rfc4122.txt` (last accessed 24/2/2019).

Lee, J.-S., Filatova, T., Ligmann-Zielinska, A., Hassani-Mahmooei, B., Stonedahl, F., Lorscheid, I., Voinov, A., Polhill, J. G., Sun, Z., and Parker, D. C. (2015). The complexities of agent-based modeling output analysis. *Journal of Artificial Societies and Social Simulation*, 18(4):4. `http://jasss.soc.surrey.ac.uk/18/4/4.html`.

Leitão, P., Mařík, V., and Vrba, P. (2013). Past, present, and future of industrial agent applications. *IEEE Transactions on Industrial Informatics*, 9(4):2360–2372.

Leondes, C. (2002). *Expert systems: the technology of knowledge management and decision making for the 21st century*. Number v. 6 in Expert Systems: The Technology of Knowledge Management and Decision Making for the 21st Century. Academic Press.

Lewis-Beck, M., Bryman, A., and Liao, T. (2004). *The SAGE Encyclopedia of Social Science Research Methods*. SAGE Publications, Thousand Oaks, California.

Liddle, S. W. (2011). Model-driven software development. In Embley and Thalheim (2011), pages 17–54.

Lilge, B. (2012). Transformation vom konzeptuellen Politikmodell in ein formales agentenbasiertes Simulationsmodell. B.S. thesis, University of Koblenz-Landau, Germany.

Lombard, M., Snyder-Duch, J., and Bracken, C. C. (2010). Practical resources for assessing and reporting intercoder reliability in content analysis research projects. `http://matthewlombard.com/reliability/` (last accessed 24/2/2019).

Lotzmann, U. (2008). TRASS - a multi-purpose agent-based simulation framework for complex traffic simulation applications. In Bazzan, A. L. C. and Klügl, F., editors, *Multi-Agent Systems for Traffic and Transportation*. IGI Global, Hershey, PA.

Lotzmann, U. (2013). D4.2 system and user documentation. SD-3: System documentation of drams tools. Deliverable 4.2, Annex SD-3 v1.0, OCOPOMO consortium. `http://www.ocopomo.eu/results/public-deliverables/system-documentation/d4.2-sd-3-system-documentation-of-drams-tools.pdf` (last accessed 24/2/2019).

Lotzmann, U. and Meyer, R. (2011a). A declarative rule-based environment for agent modelling systems. In *The Seventh Conference of the European Social Simulation Association, ESSA 2011*, Montpellier, France.

Lotzmann, U. and Meyer, R. (2011b). DRAMS - a declarative rule-based agent modelling system. In Burczynski, T., Kolodziej, J., Byrski, A., and Carvalho,

M., editors, *25th European Conference on Modelling and Simulation, ECMS 2011*, pages 77–83. SCS Europe, Krakow.

Lotzmann, U. and Meyer, R. (2013). D4.2 system and user documentation. C: User manual on policy modelling and simulation tools. Deliverable 4.2, C v1.0, OCOPOMO consortium. `http://www.ocopomo.eu/results/public-deliverables/user-manuals/d4.2-c-user-manual-on-policy-modelling-and-simulation-tools.pdf` (last accessed 24/2/2019).

Lotzmann, U. and Neumann, M. (2016). A simulation model of intra-organisational conflict regulation in the crime world. In Elsenbroich, C., Anzola, D., and Gilbert, N., editors, *Social Dimensions of Organised Crime: Modelling the Dynamics of Extortion Rackets*, pages 177–213. Springer International Publishing, Cham.

Lotzmann, U. and Neumann, M. (2017). Simulation for interpretation: A methodology for growing virtual cultures. *Journal of Artificial Societies and Social Simulation*, 20(3):13. `http://jasss.soc.surrey.ac.uk/20/3/13.html`.

Lotzmann, U., Neumann, M., and Möhring, M. (2015). From text to agents - process of developing evidence-based simulation models. In *29th European Conference on Modelling and Simulation, ECMS 2015*.

Lotzmann, U. and Wimmer, M. A. (2012). Provenance and traceability in agent-based policy simulation. In Klumpp, M., editor, *European Simulation and Modelling Conference (ESM)*, pages 202–207, Essen, Germany.

Lotzmann, U. and Wimmer, M. A. (2013a). Evidence traces for multi-agent declarative rule-based policy simulation. In *Proceedings of the 17th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2013)*, pages 115–122. IEEE Computer Society.

Lotzmann, U. and Wimmer, M. A. (2013b). Traceability in evidence-based policy simulation. In Rekdalsbakken, W., Bye, R. T., and Zhang, H., editors, *27th European Conference on Modelling and Simulation, ECMS 2013*, pages 696–702. Digitaldruck Pirrot GmbH: Dudweiler.

MacPhail, C., Khoza, N., Abler, L., and Ranganathan, M. (2016). Process guidelines for establishing intercoder reliability in qualitative studies. *Qualitative Research*, 16(2):198–212.

Magalhães, A. P., Andrade, A., and Maciel, R. S. (2016). A model driven transformation development process for model to model transformation. In *Proceedings of the 30th Brazilian Symposium on Software Engineering*, SBES '16, pages 3–12, New York, NY, USA. ACM.

Majstorovic, D., Wimmer, M. A., Lay-Yee, R., Davis, P., and Ahrweiler, P. (2015). Features and added value of simulation models using different modelling approaches supporting policy-making: A comparative analysis. In Janssen, M., Wimmer, M. A., and Deljoo, A., editors, *Policy Practice and Digital Science: Integrating Complex Systems, Social Simulation and Public Administration in Policy Research*, pages 95–123. Springer International Publishing, Cham.

Malinowski, B. (1922). *Argonauts of the Western Pacific: an account of native enterprise and adventure in the archipelagoes of Melanesian New Guinea*. Routledge & Kegan Paul, London, England.

Malleson, N., Heppenstall, A., See, L., and Evans, A. (2013). Using an agent-based crime simulation to predict the effects of urban regeneration on individual household burglary risk. *Environment and Planning B: Planning and Design*, 40(3):405–426.

Mayr, H. C., Guizzardi, G., Ma, H., and Pastor, O., editors (2017). *Conceptual Modeling - 36th International Conference, ER 2017, Valencia, Spain, November 6-9, 2017, Proceedings*, volume 10650 of *Lecture Notes in Computer Science*. Springer.

McCollough, A., Denmark, D., and Harker, D. (2014). Interliminal design: Understanding cognitive heuristics to mitigate design distortion. *FormAkademisk - Research Journal of Design and Design Education*, 7(4).

Mead, M. (1928). *Coming of age in Samoa; a psychological study of primitive youth for western civilisation*. W. Morrow & Company, New York.

Moellenkamp, S., Lamers, M., Huesmann, C., Rotter, S., Pahl-Wostl, C., Speil, K., and Pohl, W. (2010). Informal participatory platforms for adaptive management. Insights into niche-finding, collaborative design and outcomes from a participatory process in the Rhine basin. *Ecology and Society*, 15(4):41. `http://www.ecologyandsociety.org/vol15/iss4/art41/`.

Möhring, M. (1990). *MIMOSE - eine funktionale Sprache zur Beschreibung und Simulation individuellen Verhaltens in interagierenden Populationen*. PhD thesis, Universität Koblenz.

Möhring, M. (1995). Social science multilevel simulation with MIMOSE. In *Social Science Microsimulation [Dagstuhl Seminar, May, 1995]*, pages 123–137.

Moss, S. (2011). Policy modelling, open collaboration and the future of eGovernance. In *CROSSROAD Call for Contributions on "Future Internet on ICT for Governance and Policy Modelling"*. `http://www.ocopomo.eu/results/papers/MOSS-CROSSROAD.pdf` (last accessed 24/2/2019).

Moss, S. and Edmonds, B. (2005). Towards good social science. *Journal of Artificial Societies and Social Simulation*, 8(4):13. `http://jasss.soc.surrey.ac.uk/8/4/13.html`.

Moss, S., Gaylard, H., Wallis, S., and Edmonds, B. (1998). SDML: A multi-agent language for organizational modelling. *Computational Mathematical Organization Theory*, 4(1):43–69.

Müller, J. P. and Cossentino, M., editors (2013). *Agent-Oriented Software Engineering XIII - 13th International Workshop, AOSE 2012, Valencia, Spain, June 4, 2012, Revised Selected Papers*, volume 7852 of *Lecture Notes in Computer Science*. Springer.

Munroe, S., Groth, P., Jiang, S., Miles, S., Tan, V., Ibbotson, J., and Moreau, L. (2006). Overview of the provenance specification effort. Technical report. `http://eprints.soton.ac.uk/263055/1/OverviewVision.pdf` (last accessed 24/2/2019).

Nardin, L. G., Andrighetto, G., Conte, R., Székely, Á., Anzola, D., Elsenbroich, C., Lotzmann, U., Neumann, M., Punzo, V., and Troitzsch, K. G. (2016a). Simulating protection rackets: a case study of the sicilian mafia. *Autonomous Agents and Multi-Agent Systems*, 30(6):1117–1147.

Nardin, L. G., Andrighetto, G., Székely, Á., Punzo, V., and Conte, R. (2016b). An agent-based model of extortion racketeering. In Elsenbroich, C., Anzola, D., and Gilbert, N., editors, *Social Dimensions of Organised Crime: Modelling the Dynamics of Extortion Rackets*, pages 105–116. Springer International Publishing, Cham.

Nardin, L. G., Székely, Á., and Andrighetto, G. (2017). GLODERS-S: a simulator for agent-based models of criminal organisations. *Trends in Organized Crime*, 20(1):85–99.

Neumann, M. (2014). Grounded simulation. In Kaminski, B. and Koloch, G., editors, *Advances in Social Simulation*, volume 229 of *Advances in Intelligent Systems and Computing*, pages 351–359. Springer Berlin Heidelberg.

Neumann, M. and Lotzmann, U. (2014). Modelling the collapse of a criminal network. In Squazzoni, F., Baronio, F., Archetti, C., and Castellani, M., editors, *28th European Conference on Modelling and Simulation, ECMS 2014*, pages 765–771. European Council for Modeling and Simulation.

Neumann, M. and Lotzmann, U. (2016a). Eine Herrschaft des Terrors: Gewalteskalation in illegalen Organisationen. In Equit, C., Grönemeyer, A., and Schmidt, O., editors, *Situationen der Gewalt*, Advances in Intelligent Systems and Computing. VS-Verlag Wiesbaden.

Neumann, M. and Lotzmann, U. (2016b). Sanction recognition: A simulation model of extended normative reasoning. In *Proceedings of the 10th International workshop on Normative Multiagent Systems, NorMAS 2016*.

Neumann, M. and Lotzmann, U. (2016c). Text data and computational qualitative analysis. In Elsenbroich, C., Anzola, D., and Gilbert, N., editors, *Social Dimensions of Organised Crime: Modelling the Dynamics of Extortion Rackets*, pages 155–176. Springer International Publishing, Cham.

Neumann, M., Lotzmann, U., and Troitzsch, K. G. (2017). Mafia war: simulating conflict escalation in criminal organizations. *Trends in Organized Crime*, 20(1):139–178.

Neumann, M. and Sartor, N. (2016). A semantic network analysis of laundering drug money. *Journal of Tax Administration*, 2(1):73–94.

Nguyen-Duc, M. and Drogoul, A. (2007). Using computational agents to design participatory social simulations. *Journal of Artificial Societies and Social Simulation*, 10(4):5. `http://jasss.soc.surrey.ac.uk/10/4/5.html`.

North, M., Collier, N., Ozik, J., Tatara, E., Macal, C., Bragen, M., and Sydelko, P. (2013). Complex adaptive systems modeling with Repast Simphony. *Complex Adaptive Systems Modeling*, 1(1):3. `http://www.casmodeling.com/content/1/1/3`.

North, M. J., Collier, N. T., and Vos, J. R. (2006). Experiences creating three implementations of the repast agent modeling toolkit. *ACM Trans. Model. Comput. Simul.*, 16(1):1–25.

Object Management Group (OMG) (2014). MDA Guide revision 2.0. `https://www.omg.org/cgi-bin/doc?ormsc/14-06-01` (last accessed 24/2/2019).

OCOPOMO consortium (2012). OCOPOMO Glossary. `http://www.ocopomo.eu/results/glossary` (last accessed 24/2/2019).

Otte, E. and Rousseau, R. (2002). Social network analysis: a powerful strategy, also for the information sciences. *Journal of Information Science*, 28(6):441–453.

Özbaş, B., Özgün, O., and Barlas, Y. (2014). Modeling and simulation of the endogenous dynamics of housing market cycles. *Journal of Artificial Societies and Social Simulation*, 17(1):19. `http://jasss.soc.surrey.ac.uk/17/1/19.html`.

Pastor, O. (2017). Model-driven development in practice: From requirements to code. In Steffen, B., Baier, C., van den Brand, M., Eder, J., Hinchey, M., and Margaria, T., editors, *SOFSEM 2017: Theory and Practice of Computer Science*, pages 405–410, Cham. Springer International Publishing.

Picek, R. and Strahonja, V. (2007). Model driven development - future or failure of software development. In *In IIS'07: 18th International Conference on Information and Intelligent Systems*.

Polhill, J. G., Sutherland, L., and Gotts, N. M. (2010). Using qualitative evidence to enhance an agent-based modelling system for studying land use change. *Journal of Artificial Societies and Social Simulation*, 13(2):10. `http://jasss.soc.surrey.ac.uk/13/2/10.html`.

Railsback, S., Ayllón, D., Berger, U., Grimm, V., Lytinen, S., Sheppard, C., and Thiele, J. (2017). Improving execution speed of models implemented in NetLogo. *Journal of Artificial Societies and Social Simulation*, 20(1):3. `http://jasss.soc.surrey.ac.uk/20/1/3.html`.

Raper, J. (1993). Geographical information systems. *Progress in Physical Geography: Earth and Environment*, 17(4):493–502.

Reeves, J. W. (2005). Code as design: Three essays. `http://www.developerdotstar.com/mag/articles/reeves_design_main.html` (last accessed 24/2/2019).

Reinhardt, O., Hilton, J., Warnke, T., Bijak, J., and Uhrmacher, A. M. (2018). Streamlining simulation experiments with agent-based models in demography. *Journal of Artificial Societies and Social Simulation*, 21(3):9. `http://jasss.soc.surrey.ac.uk/21/3/9.html`.

Robinson, S. (2008). Conceptual modelling for simulation part I: Definition and requirements. *The Journal of the Operational Research Society*, 59(3):278–290.

Robinson, S., Arbez, G., Birta, L. G., Tolk, A., and Wagner, G. (2015). Conceptual modeling: Definition, purpose and benefits. In *2015 Winter Simulation Conference (WSC)*, pages 2812–2826.

Sabater-Mir, J., Paolucci, M., and Conte, R. (2006). Repage: REPutation and imAGE among limited autonomous partners. *Journal of Artificial Societies and Social Simulation*, 9(2):3. `http://jasss.soc.surrey.ac.uk/9/2/3.html`.

Salgado, M., Marchione, E., and Gilbert, N. (2014). Analysing differential school effectiveness through multilevel and agent-based modelling. *Journal of Artificial Societies and Social Simulation*, 17(4):3. `http://jasss.soc.surrey.ac.uk/17/4/3.html`.

Sandberg, M. (2011). Soft power, world system dynamics, and democratization: A bass model of democracy diffusion 1800-2000. *Journal of Artificial Societies and Social Simulation*, 14(1):4. `http://jasss.soc.surrey.ac.uk/14/1/4.html`.

Sartor, N. (2015). Eine Netzanalyse der Organisationsstrukturen im Bereich Geldwäsche. master thesis, Universität Koblenz-Landau, Universitätsbibliothek.

Sartor, N., Kaspers, T., Neumann, M., Lotzmann, U., Möhring, M., and Troitzsch, K. G. (2014). D 2.3 consolidated database of ers documentation. Deliverable 2.3. FP7 project GLODERS (Global Dynamics of Extortion Racket Systems). Deliverable. `http://www.gloders.eu/images/Deliverables/GLODERS_D2-3.pdf` (last accessed 24/2/2019).

Schelling, T. (1969). Models of segregation. *American Economic Review*, 59(2):488–493.

Scherer, S. (2016). *Towards an E-Participation Architecture Framework (EPArt-Framework)*. PhD thesis, Universität Koblenz-Landau, Universitätsbibliothek.

Scherer, S., Lotzmann, U., Wimmer, M. A., Furdik, K., Mach, M., Sabol, T., Butka, P., Bednar, P., Moss, S., Pinotti, D., Kacprzyk, M., Smatana, P., and Roszczynska-Kurasinska, M. (2013a). D8.1 manual of the methodology for process development and guide to policy modelling toolbox. Deliverable 8.1 v1.0, OCOPOMO consortium. `http://www.ocopomo.eu/results/public-deliverables/d-8.1-manual-of-the-methodology-for-process-development-and-guide-to-policy-modelling-toolbox` (last accessed 24/2/2019).

Scherer, S., Wimmer, M., Lotzmann, U., Moss, S., and Pinotti, D. (2015). Evidence based and conceptual model driven approach for agent-based policy modelling. *Journal of Artificial Societies and Social Simulation*, 18(3):14. `http://jasss.soc.surrey.ac.uk/18/3/14.html`.

Scherer, S., Wimmer, M. A., and Markisic, S. (2013b). Bridging narrative scenario texts and formal policy modeling through conceptual policy modeling. *Artificial Intelligence and Law*, 21(4):455–484.

Schwaber, K. (2004). *Agile Project Management With Scrum*. Microsoft Press, Redmond, WA, USA.

Sherrell, L. (2013). Evolutionary prototyping. In Runehov, A. L. C. and Oviedo, L., editors, *Encyclopedia of Sciences and Religions*, pages 803–803. Springer Netherlands, Dordrecht.

Siebers, P.-O. and Davidsson, P. (2015). Engineering agent-based social simulations: An introduction. *Journal of Artificial Societies and Social Simulation*, 18(3):13. `http://jasss.soc.surrey.ac.uk/18/3/13.html`.

Silverman, E., Bijak, J., Courgeau, D., and Franck, R. (2014). Advancing social simulation: Lessons from demography. In Sayama, H., Rieffel, J., Risi, S., Doursat, R., and Lipson, H., editors, *Artificial Life 14*, pages 384–391.

Sokolowski, J. A. and Banks, C. M. (2009). *Principles of Modeling and Simulation: A Multidisciplinary Approach*. John Wiley & Sons Inc., Hoboken, NJ.

Sokolowski, J. A. and Banks, C. M. (2010). *Modeling and Simulation Fundamentals: Theoretical Underpinnings and Practical Domains*. John Wiley & Sons Inc., Hoboken, NJ.

Sommerville, I. (2010). *Software Engineering*. Addison-Wesley Publishing Company, USA, 9th edition.

Squazzoni, F., Jager, W., and Edmonds, B. (2014). Social simulation in the social sciences: A brief overview. *Social Science Computer Review*, 32(3):279–294.

Subramaniam, V. (2014). *Functional Programming in Java: Harnessing the Power of Java 8 Lambda Expressions*. Pragmatic programmers. Pragmatic Bookshelf.

Takeuchi, H. and Nonaka, I. (1986). The new new product development game. *Harvard Business Review*.

Taylor, Q. and Giraud-Carrier, C. (2010). Applications of data mining in software engineering. *Int. J. Data Anal. Tech. Strateg.*, 2(3):243–257.

Taylor, R. (2003). *Agent-Based Modelling Incorporating Qualitative and Quantitative Methods: A Case Study Investigating the Impact of E-commerce upon the Value Chain*. PhD thesis, Manchester Metropolitan University, Manchester, UK.

Thalheim, B. (2011). The art of conceptual modelling. In *Information Modelling and Knowledge Bases XXIII, 21st European-Japanese Conference on Information Modelling and Knowledge Bases (EJC 2011), Tallinn, Estonia, June, 6-10, 2011*, pages 149–168.

Troitzsch, K. G. (2004). Validating simulation models. In *Proceedings of 18th European Simulation Multiconference on Networked Simulation and Simulation Networks, SCS Publishing House*, pages 265–270.

Troitzsch, K. G. (2016a). Extortion rackets: An event-oriented model of interventions. In Elsenbroich, C., Anzola, D., and Gilbert, N., editors, *Social Dimensions of Organised Crime: Modelling the Dynamics of Extortion Rackets*, pages 117–131. Springer International Publishing, Cham.

Troitzsch, K. G. (2016b). Survey data and computational qualitative analysis. In Elsenbroich, C., Anzola, D., and Gilbert, N., editors, *Social Dimensions of Organised Crime: Modelling the Dynamics of Extortion Rackets*, pages 133–151. Springer International Publishing, Cham.

Trujillo, J., Davis, K. C., Du, X., Li, Z., Ling, T. W., Li, G., and Lee, M., editors (2018). *Conceptual Modeling - 37th International Conference, ER 2018, Xi'an, China, October 22-25, 2018, Proceedings*, volume 11157 of *Lecture Notes in Computer Science*. Springer.

Van Putten, C. and Neumann, M. (2018). Growing criminal culture: Narrative simulation of conflicts in a criminal group. `https://www.researchgate.net/publication/325194641_Growing_Criminal_Culture_Narrative_Simulation_of_Conflicts_in_a_Criminal_Group` (last accessed 24/2/2019).

Venturini, T., Jensen, P., and Latour, B. (2015). Fill in the gap: A new alliance for social and natural sciences. *Journal of Artificial Societies and Social Simulation*, 18(2):11. `http://jasss.soc.surrey.ac.uk/18/2/11.html`.

Voinov, A. and Bousquet, F. (2010). Position paper: Modelling with stakeholders. *Environ. Model. Softw.*, 25(11):1268–1281.

Waldherr, A. and Wijermans, N. (2013). Communicating social simulation models to sceptical minds. *Journal of Artificial Societies and Social Simulation*, 16(4):13. `http://jasss.soc.surrey.ac.uk/16/4/13.html`.

Weber, M. (1968). Objektive Möglichkeit und adäquate Verursachung in der historischen Kausalbetrachtung. In Winckelmann, J., editor, *Gesammelte Aufsätze zur Wissenschaftslehre*, pages 266–290. Mohr Siebeck, Tübingen, 3rd edition.

Wiegers, K. E. and Beatty, J. (2013). *Software Requirements 3*. Microsoft Press, Redmond, WA, USA.

Wimmer, M. A. (2011). Open government in policy development: From collaborative scenario texts to formal policy models. In Natarajan, R. and Ojo, A., editors, *Distributed Computing and Internet Technology*, volume 6536 of *Lecture Notes in Computer Science*, pages 76–91. Springer Berlin / Heidelberg.

Wimmer, M. A. and Bicking, M. (2011). Collaborative scenario building for policy modelling. In Ghoneim, A., Veerakkody, V., and Kamal, M., editors, *tGov2011*, London, UK. Brunel University.

Wimmer, M. A., Furdik, K., Bicking, M., Mach, M., Sabol, T., and Butka, P. (2012a). Open collaboration in policy development: Concept and architecture to integrate scenario development and formal policy modelling. In

Charalabidis, Y. and Koussouris, S., editors, *Empowering Open and Collaborative Governance*, pages 119–219. Springer Verlag, Berlin.

Wimmer, M. A., Scherer, S., Moss, S., and Bicking, M. (2012b). Method and tools to support stakeholder engagement in policy development: The OCOPOMO project. *IJEGR*, 8(3):98–119.

Wooldridge, M. and Jennings, N. R. (1995). Intelligent agents: theory and practice. *The Knowledge Engineering Review*, 10(2):115–152.

Wooldridge, M. J. (2002). *Introduction to multiagent systems*. Wiley.

Yang, L. and Gilbert, N. (2008). Getting away from numbers: Using qualitative observation for agent-based modeling. *Advances in Complex Systems*, 11(2):175–185.

Zeigler, B. P. (1984). *Theory of Modelling and Simulation*. Krieger Publishing Co., Inc., Melbourne, FL, USA.

Zeigler, B. P., Praehofer, H., and Kim, T. G. (2000). *Theory of Modeling and Simulation, Second Edition*. Academic Press, London, UK.

Curriculum Vitae

ULF LOTZMANN

PERSONAL INFORMATION

Born in Hoyerswerda, 14 September 1974

email ulf@lotzmann.net

EDUCATION

2009–2019 Doctoral Studies in Modelling and Simulation

University of Koblenz-Landau, Faculty of Computer Science, Department of IS Research, Koblenz, Germany
Candidate for the Degree of Doctor in Natural Sciences (Dr. rer. nat.)

1994–2006 Diploma Programme in Computer Science

University of Koblenz-Landau, Faculty of Computer Science, Department of IS Research, Koblenz, Germany
Diploma in Computer Science (Dipl.-Inform.)

WORK EXPERIENCE

2006–Present Research Associate

University of Koblenz-Landau, Faculty of Computer Science, Department of IS Research, Koblenz, Germany

2000–2006 Student Assistant

University of Koblenz-Landau, Faculty of Computer Science, Department of IS Research, Koblenz, Germany

1998–2000 Student Assistant

University of Koblenz-Landau, Faculty of Educational Sciences, Koblenz, Germany

February 27, 2019