UNIVERSITÄT KOBLENZ · LANDAU
Fachbereich 4:
Institut für Wirtschafts- und Verwaltungsinformatik

Micro-agents revisited: A Modern Reimplementation of the Micro-agent Layer of the Otago Agent Platform (OPAL)

Masterarbeit

zur Erlangung des Grades eines
Master of Science
im Studiengang Wirtschaftsinformatik

vorgelegt von

Christopher Frantz

Betreuer:	Prof. Dr. Martin K. Purvis, University of Otago
	Prof. Dr. Klaus G. Troitzsch, Institut für Wirtschafts- und Verwaltungsinformatik, Fachbereich Informatik
	Dr. Mariusz Nowostawski, University of Otago
Erstgutachter:	Prof. Dr. Martin K. Purvis, University of Otago
Zweitgutachter:	Prof. Dr. Klaus G. Troitzsch, Institut für Wirtschafts- und Verwaltungsinformatik, Fachbereich Informatik

Koblenz, im November 2010

ERKLÄRUNG

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden.
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.

Ort, Datum	Unterschrift

Abstract

Multi-agent systems are a mature approach to model complex software systems by means of Agent-Oriented Software Engineering (AOSE). However, their application is not widely accepted in mainstream software engineering.
Parallel to this the interdisciplinary field of Agent-based Social Simulation (ABSS) finds increasing recognition beyond the purely academic realm which starts to draw attention from the mainstream of agent researchers.
This work analyzes factors to improve the uptake of AOSE as well as characteristics which separate the two fields AOSE and ABSS to understand their gap. Based on the efficiency-oriented micro-agent concept of the Otago Agent Platform (OPAL) we have constructed a new modern and self-contained micro-agent platform called μ^2. The design takes technological trends into account and integrates representative technologies, such as the functionally-inspired JVM language Clojure (with its Transactional Memory), asynchronous message passing frameworks and the mobile application platform Android.
The mobile version of the platform shows an innovative approach to allow direct interaction between Android application components and micro-agents by mapping their related internal communication mechanisms. This empowers micro-agents to exploit virtually any capability of mobile devices for intelligent agent-based applications, robotics or simply act as a distributed middleware.
Additionally, relevant platform components for the support of social simulations are identified and partially implemented. To show the usability of the platform for simulation purposes an interaction-centric scenario representing group shaping processes in a multi-cultural context is provided. The scenario is based on Hofstede's concept of 'Cultural Dimensions'. It does not only confirm the applicability of the platform for simulations but also reveals interesting patterns for culturally augmented in- and out-group agents.
This explorative research advocates the potential of micro-agents as a powerful general system modelling mechanism while bridging the convergence between mobile and desktop systems. The results stimulate future work on the micro-agent concept itself, the suggested platform and the deeper exploration of mechanisms for seemless interaction of micro-agents with mobile environments. Last but not least the further elaboration of the simulation model as well as its use to augment intelligent agents with cultural aspects offer promising perspectives for future research.

Zusammenfassung

Die Verwendung von Multi-Agenten-Systemen in Verbindung mit Agenten-orientiertem Software Engineering stellt mittlerweile einen ausgereiften Ansatz zur Modellierung komplexer Systeme dar. Allerdings finden Multi-Agenten-Systeme im weiteren Feld der Softwareentwicklung bisher nur schwachen Anklang.
Im Kontrast dazu erregt die Anerkennung des interdisziplinären Feldes der sozialwissenschaftlichen Simulation über akademische Grenzen hinaus zunehmend die Aufmerksamkeit des 'Mainstream' der Agentenforscher. Dies nimmt sich diese Arbeit zum Anlass, die Verwendung von Multi-Agenten-Systemen im weiteren Feld der Software-Entwicklung voranzutreiben und gleichzeitig die unterschiedlichen Schwerpunkte von sozialwissenschaftlicher Simulation und Multi-Agenten-Systemen zu analysieren. Im Zuge dessen bietet sich die Reimplementierung des effizienzorientierten Micro-Agenten-Konzepts, welches den Kern der Otago Agent Platform (OPAL) bildet, an. Ergebnis ist eine eigenständige Micro-Agenten-Plattform, μ^2, die zahlreiche konzeptionelle wie technische Innovationen bietet. Dazu gehören die Integration der funktional orientierten Programmiersprache Clojure sowie die Implementation asynchroner Nachrichtenübermittlung. Als weiterer Aspekt wird die zunehmend populäre mobile Anwendungsplattform Android berücksichtigt.
Über die reine Portierung der Agentenplattform hinaus weist Android Merkmale von Multi-Agenten-Systemen auf. Die Verbindung der Kommunikationsmechanismen von Android und Micro-Agenten-Plattform erweitert das Funktionsspektrum auf beiden Seiten: Micro-Agenten können unmittelbar auf beliebige Android-Funktionalität zugreifen, während Android die Micro-Agenten-Plattform als netzwerkweite Kommunikationsmiddleware verwenden kann. Potentiale dieser symbiotischen Beziehung erstrecken sich darüber hinaus in die Bereiche der Entwicklung 'intelligenter' mobiler Anwendungen sowie Robotik.
Weiteres Ziel der Arbeit ist die Bereitstellung von Erweiterungen der Plattform, die eine Verwendung im Kontext sozialwissenschaftlicher Simulation ermöglichen. Das aus Softwareentwicklungssicht schwache, jedoch aus Simulationssicht relativ mächtige Micro-Agenten-Konzept zeigt seine Qualitäten im Kontext kommunikationsintensiver Szenarien. Um dies zu realisieren, wird ein Simulationszenario basierend auf Hofstede's Cultural Dimensions konstruiert, welches die Gruppenbildung interkultureller Individuen simuliert. Neben dem Nachweis der Anwendbarkeit der Plattform für Simulationsszenarien zeigen sich interessante Muster in und außerhalb der gebildeten Gruppen.

Insgesamt zeigt diese explorative Arbeit ein breites Potential zur Verwendung von Agenten- respektive Micro-Agenten-Prinzipien auf. Dies reicht von Agenten-orientierter Modellierung bis zum Potential, die Konvergenz mobiler und stationärer Anwendungen voranzutreiben. Darüber hinaus bieten sich weitere Verbesserungspotentiale am Micro-Agenten-Konzept sowie der Plattform selbst, ebenso wie eine intensivere Untersuchung der direkten Interaktion von Micro-Agenten mit mobilen Umgebungen an. Unabhängig davon bietet das entwickelte Simulationsmodell einen Ansatz für die Verwendung kultureller Aspekte im Kontext intelligenter Agenten.

Acknowledgements

I want to express my deepest gratitude to my supervisors Professor Dr. Martin K. Purvis and Professor Dr. Klaus G. Troitzsch whose joint supervision effort and tireless support made this thesis 'at the antipodes' possible.
I also want to express my deepest gratitude to my supervisor Dr. Mariusz Nowostawski whose feedback was crucial for the outcome of the work.

Table of Contents

List of Figures

List of Tables

List of Listings

Abbreviations

ABSS	Agent-Based Social Simulation
ACID	Atomicity, Isolation, Consistency, Durability
ACL	Agent Communication Languages
ACO	Ant Colony Optimization
AGR	Agent-Group-Role (model)
AI	Artificial Intelligence
AID	Agent Identifier
ALL	Agent Logic Layer
AMS	Agent Management System
AO	Agent-Orientation
AOP	Agent-Oriented Programming
AOSE	Agent-Oriented Software Engineering
BDI	Belief-Desire-Intention (model)
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
CSV	Comma-separated values
DAI	Distributed Artificial Intelligence
DBMS	Database Management System
DF	Directory Facilitator
DIS	Distributed Information Systems
DPS	Distributed Problem Solving
FIPA	Foundation for Intelligent Physical Agents
GST	General Systems Theory
HTTP	Hypertext Transfer Protocol

IDV	Individualism Index (Cultural dimension)
IIOP	Internet Inter-ORB Protocol
J2ME	Java 2 MicroEdition
JAS	Java Agent Services
JCP	Java Community Process
JIT	Just-in-Time
JSR	Java Specification Request
JVM	Java Virtual Machine
KIF	Knowledge Interchange Format
KQML	Knowledge and Query Manipulation Language
LCG	Linear Congruential Generator
LISP	List Processing
MABS	Multi-Agent-Based Simulation
MAD	Modelling, Analysis and Design (methodology)
MAS	Multi-Agent Systems
MASIF	Mobile Agent System Interoperability Facility
MOA	Micro-agents on Android
MRL	Message Routing (& Platform Management) Layer
MTL	Message Transport (& Platform Runtime) Layer
MTS	Message Transport System
M&S	Modelling and Simulation
NZDIS	New Zealand Distributed Information Systems
OCC	Ortony Clore Collins (theory)
OMG	Object Management Group
OO	Object-Orientation
OOAD	Object-oriented Analysis and Design
OPAL	Otago Agent Platform
ORB	Object Request Broker
OS	Operating System
PDI	Power Distance Index (Cultural dimension)
PDU	Protocol Data Unit

PRS	Procedural Reasoning System
REPL	Read-Eval-Print-Loop
RPC	Remote Procedure Call
SMS	Short Message Service
STM	Software Transactional Memory
UAI	Uncertainty Avoidance Index (Cultural dimension)
URI	Uniform Resource Identifier

Chapter 1

Introduction

1.1 Background and Motivation

The application of agent-based software concepts receives a practical interest which is going beyond the numerous provisions of agent development frameworks. Application areas for agent-based technology in the industrial context include Intelligent Manufacturing, Enterprise Integration (see [MM05] and [SHYN06]) and Logistics[1] along with numerous specialized applications (e.g. intelligent power management, operation management (see [PTB+06] for an overview)).

But yet agent-based software is not part of the general software engineering toolbox (as acknowledged by Georgeff [Geo09] and Winikoff [Win09]). This thesis suggests that the concentration of academic research on rather heavy-weight 'intelligent' agent technology in the context of Agent-Oriented Software Engineering (AOSE) is a partial reason for this as relevant areas such as performance are neglected.

One application field whose approach implicitly supports this view is the area of simulation, in particular Agent-Based Social Simulation (ABSS). An indicator for the increasing importance of this field is an analysis undertaken by Michel et al. [MFD09] showing that a fraction of 35% of all papers of the 2007 Conference on Autonomous Agents and Multiagent Systems (AAMAS), the key conference

[1] An example for this field and significant productivity advantages achieved by using agent-based software is provided by Benfield et al. [BHG06].

in the field, was concerned with simulation aspects[2]. The establishment of agent-based simulation as a permanent topic of interest in this conference series supports this position. Additionally, agent-based simulation of social aspects, and in particular economic problems (such as source and impact of the recent economic crisis) receives increasing attention from policy makers and general public (see 'Economist' [Eco] and Ball [Bal10]) – reason to suggest that agent-based simulation is successful with regards to a general acceptance.

Independent from the areas mentioned above – which are interlinked by their focus on agents – information and communication technology continuously evolves with new or rediscovered trends (e.g. processor architectures, programming languages). One of those is the increasing number of processing cores even on low end desktop machines, making the consideration of parallelization in software mandatory to lever the hidden performance potential (as stated by Sutter in 2005 [Sut05]). In strong relation to this, historically older programming paradigms such as functional programming are revived, mainly because they address key drawbacks of the yet dominant object-oriented programming paradigm[3], in particular their stronger focus on immutibility which eases the handling of concurrency.

Another ongoing trend is the continuous convergence of mobile devices (and their markets) with desktop systems, now in the shape of smart phones and the revived notion of electronic 'pads'[4]. In fact their boom could potentially outperform the desktop market by 2013 (see announcement of Google CEO Eric Schmidt [Ber]). Particular candidates of concern in this context are devices running the Android operation system[5]. Apart from the market perspective, Android itself implicitly shows relations to multi-agent systems suggesting the uptake of agent-based technologies on mobile systems 'through the backdoor' without explicit acknowledgement.

[2]Here it should be mentioned that those figures refer to agent-based simulation in general, not only social simulation.

[3]An indicator for this is the TIOBE Programming Community Index [B.V] indicating a dominating fraction of more than 50% for object-oriented languages.

[4]Those electronic pads in fact revive a successless promotion of 'tablet PCs' in the early 2000's, mainly caused by limited performance at that time.

[5]Although the number of sold units increased for nearly all smart phone vendors, Android-based systems (across various vendors) gained an increase of 886% in Q2 2010 as compared to the same time period in 2009 (see Canalys [Can]).

As all these aspects (multi-core processing, programming languages, mobile application platforms) impinge on software engineering in general, their combined consideration is of interest to promote the uptake of agent-based principles. A candidate to realize this is the micro-agent layer of the Otago Agent Platform (OPAL) [NBPC01], a multi-level agent platform developed at the University of Otago, which supports coarse-grained (i.e. considerably powerful and standard-oriented but inefficient) agents as well as efficiency[6]-motivated micro-agents. Particularly the concept of micro-agents takes a pragmatic approach towards Agent-Oriented Software Engineering, which makes it suitable for a consistent 'modelling in agents' without significant performance penalties in comparison to other programming paradigms which use simpler modelling entities (such as Object-Orientation). Beyond its strengths the micro-agent layer implementation has various limitations such as the delegation of concurrency handling to the user and lack of network support. As an approach to provide modern Agent-Oriented Software Engineering a new micro-agent platform is suggested to realize the synergy effects of 'updating' the micro-agent concept and its implementation with a feature extension using recently emerged technologies.

The overall result shall not only show that the discussed aspects can be combined but also realize reciprocal advantages of using micro-agents to facilitate software engineering from a consequently agent-oriented perspective. The application developer (= platform user) shall be freed from any low level concern such as thread handling or networking as well as the interoperation with mobile devices.

Along with this the potential application field of micro-agents (which are rooted in the area of agent-oriented software engineering) should be extended to the more interdisciplinary field of social simulations. In consequence a scenario to provide insight into group shaping behaviour among culturally augmented agents is suggested. In turnaround this serves as 'proof of concept' for the platform's reliability – with regards to simulation requirements such as replicability – as well as robustness – with regards to handling of massively concurrent asynchronous agent interactions.

[6]Efficiency in the context of micro-agents consistently refers to 'performance efficiency'.

Figure 1.1 summarizes the influence factors for a reimplementation of the micro-agent framework.

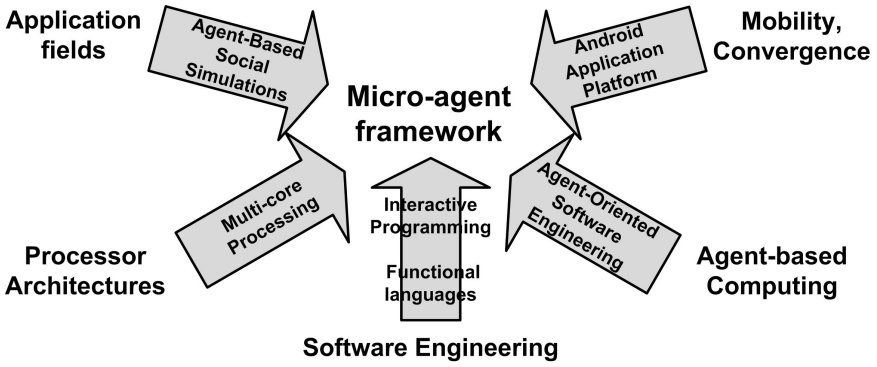


FIGURE 1.1: Influence factors for the micro-agent framework reimplementation

1.2 Outline of the thesis

The thesis describing this approach is structured as follows:

The next chapter provides an overview on terminological and conceptual foundations, such as the wide range of agent understandings and a short overview on related agent architectures. Rooted in the imprecise agent concept, a schema for an extended understanding of agency notions is suggested (section 2.1.3) and used as a communication vehicle for the remaining chapters.

A further part of the second chapter is concerned with the concept of multi-agent system and argues why those are more than 'just a bunch of agents'. To do this, first an introduction on relevant concepts of the systems theory is provided. Those are related to the principles of multi-agent systems, seeking to clarify the potential of multi-agent systems as an appropriate paradigm to model complex systems. Finally selected specifications with relevance to multi-agent systems in general are briefly introduced. Although they are not of key concern for this work, they support the argument for micro-agents.

The third chapter introduces two research fields of concern for multi-agent systems, namely Agent-Oriented Software Engineering and Agent-based Social Simulation. Motivations and history of both are briefly introduced and key aspects discussed. Problems of each area are highlighted to give a better understanding of existing problems and future directions. The analysis of the differing understanding and view on multi-agent systems (and platforms in particular) in both disciplines clarifies relevant aspects when designing multi-agent systems for either one or both

disciplines (section 3.2.3).

The fourth chapter introduces the problem of concurrency handling, its importance in the context of modern software engineering, and structures possible approaches to tackle it. Concurrency models of the LISP dialect Clojure and the mobile application platform Android are discussed with regards to their relevance for the micro-agent platform reimplementation. Additionally a comparison of several Java-based Message Passing frameworks is presented in order to clarify the trade-offs lying in the use of this concurrency handling mechanism.

The fifth, and considerably extensive, chapter finally introduces the micro-agent concept along with the existing implementation. The limitations of the current implementation are discussed and requirements for a reimplementation are outlined. The design for a successor is suggested and selected implementation decisions are described. This section (section 5.2.2) also discusses the performance[7] of the reimplemented platform. Additionally the extensions to the core implementation, such as Clojure as alternative agent implementation language and a fair scheduler for the use with simulations, are briefly discussed. A last important aspect of this chapter is the introduction of the mobile version of the reimplemented platform, targeting the Android platform. Apart from a simple port to the mobile operating system, the work with the reimplemented micro-agents and Android provides an interesting outcome with regards to the direct interoperation of both – in the spirit of open systems and mobile Agent-Oriented Software Engineering.

The sixth chapter is dedicated to a simulation case showing the applicability of the (desktop version of the) platform in the context of simulations. The simulation case models a group-shaping process using Hofstede's 'Cultural Dimensions' [Hof01] in order to understand human grouping behaviour by means of culturally augmented agent societies. The modelling process is described and the outcome as well as the general value of both model and platform are discussed.

Finally, the conclusion recalls the achievements of this work, lists the limitations of all covered aspects and suggests areas for future work.

[7] Performance in this context refers to message throughput.

Chapter 2

Terminological and Conceptual Foundations

2.1 The Agent concept

Before introducing the specific research fields of Agent-Oriented Software Engineering (AOSE) and Agent-Based Social Simulation (ABSS), the elementary concepts for this work are discussed.

The (software) agent concept is defined in manifold attempts which managed the trade-off of abstraction and tractability differently. This section does not yield to provide a final solution to this problem but rather point out the potential dimensions of definition approaches.

2.1.1 Definitions and Notions

The direct translation from the Latin verb *agere* – to act, lead, do – does not provide a distinct understanding of what (software) agents actually are. Earlier circumlocutions, such as "digital sister-in-law" by Negroponte [Neg97], indicate that agents find their use in socio-technical systems - in whatever actual distance from the user. Ascriptive approaches which compare agents to objects include their understanding as *Objects on Steriods* (see van Dyke Parunak [vDP00]). Apart from the sense of humour this does not add any tractable precision.

Before looking at descriptive approaches to cover the agent concept the fields from which (Multi-)Agent systems originate, should be clarified: Originally shaped in

the hope to develop entities with abilities similar to human beings, they derive from the Artificial Intelligence (AI) community which puts a focus on the internals of a typically single agent. From starvation of further progress (during what was later named 'AI Winter' [Cre93]) and the successive relevance of Distributed Information Systems (DIS) the field of Multi-Agent Systems (MAS) emerged as a specialization of Distributed Artificial Intelligence (DAI) (as illustrated by Bond and Gasser [BG88]). It rediscovers elements from AI and combines those with direct communication mechanisms to switch to an interaction-centric decentralized approach in system construction. This opposes the focus on central coordination and problem decomposition as the case with Distributed Problem Solving (DPS) and DAI in general[1]. However, the notions and heritage of different research groups cannot be ignored – a basic reason for the widely differing interpretation of the agent concept.

A minimal consensus is the acceptance of viewing Multi-Agent systems as intentional systems as suggested by Dennett [Den87] who introduces several predictive stances of human beings towards systems. He describes the physical, design and intentional stance, starting from the view of a system as explainable by *physical characteristics and constraints* – as is the case with an object falling towards the ground – which he calls *physical stance*. The *design stance* suggests an understanding of a system based on *what it is designed for* respectively its functions. The design stance describes a higher level of abstraction than the physical stance – its functions are intuitively linked to the conceptual understanding (e.g. functions/mechanisms of an automobile). The *intentional stance* suggests that humans attribute systems whose functions they do not understand – or systems which are characterized by non-determinism – *mental capabilities with a concept of rational action.*

The intentional stance is the philosophical basis which has driven key steps in the development towards *Agent-Oriented Software Engineering* (AOSE)[2] which will be shown in the following discussion of descriptive approaches to define the concept

[1] A summarizing overview on key metaphors of different specializations is provided by Huhns [Huh09].

[2] AOSE itself will be discussed in section 3.

'agent'. To avoid misunderstanding in the upcoming paragraph and throughout this text it should be emphasized that all discussed definitions and uses of the word agent in fact refer to the concept of *software* agents.

A useful bottom-line in the understanding is a definition which Shoham considers common sense in the Artificial Intelligence (AI) community (in 1997):

"[An agent is] an entity that functions continuously and autonomously in an environment in which other processes take place and other agents exist." [Sho97] As such continuous runtime and autonomy are key features along with the concept of an environment.

Another definition which has found popularity in the context of the arising Agent-Oriented Software Engineering (AOSE) and introduces more agent properties is shaped by Jennings and Wooldridge:

"[An] agent is an encapsulated computer system that is situated in some environment, and that is capable of flexible, autonomous action in that environment in order to meet its design objectives." [JW00]

Although this definition limits what could be understood as an agent it still is at the bottom line; Singh and Huhns – certainly partially related to their interest in agent communication – consider the "capability of interacting with other agents at the social or communicative level" (see Singh [SH99] in combination with Huhns [HS97], [HS98]) as a must.

A definition of a conceptually more powerful agent is finally taken from Shoham[3]:

"An agent is an entity whose state is viewed as consisting of mental components such as beliefs, capabilities, choices and commitments." [Sho97]

Although it does not explicitly mention the aforementioned properties of an agent, its context implicitly does. This definition needs to be seen in the context of Shoham's concept of Agent-Oriented Programming (AOP) which suggests the 'programming of mental states', and as such is a clear support for the intentional stance.[4]

With the addition of the last definition the possible degrees of agent notions

[3]Shoham's earlier definition is his interpretation of the predominant understanding within the community – which he obviously only partially shares.

[4]The idea of AOP will be reflected at a later stage.

are covered. Researchers typically group those into weak and strong notions of agency whose discriminating criterion can be more or less explicit. Explicit approaches suggest the existence of mentalistic or even emotional qualities (see Shoham [Sho97] as well as Wooldridge and Jennings [WJ95]). Moulin and Chaibdraa [MCd96] further require explicit reasoning capabilities to qualify as what they define as an *Intentional Agent*. Other approaches for the distinction of notions are guided by the external perception of agents (such as Wooldridge, Jennings and Kinny [WJK00]). Agents of weak notion are perceived as black box, defined by a strictly behaviouristic perspective only concentrating on stimuli and response of an agent entity[5]. Strong notions are characterized by a white box understanding which gives more insight respectively understanding of internal processes. The latter can be considered more granular as it allows the idea to see agents as grey boxes, allowing to describe at least some of the internals.

In essence, as with actual definitions, classifications offer a similarly large space hampering a common ground. Franklin and Graesser [FG96] as well as Etzioni and Weld [EW95] provide notable discussions of numerous agent definitions and classifications which serve as a useful source of candidate agent properties.

The following listing structures the numerous partially considerably overlapping agent properties and differing terms into core properties. The properties mentioned in the listing are briefly ordered by relevance, starting with properties of compulsory character towards advanced features pointing towards the strong notion of agency:

- Reactivity – Reactive behaviour is the absolute minimum to expect from agents. However, the term itself suggests a broad interpretation which includes all communicating computational entities – and likely to be the reason why it is hardly explicitly mentioned in agent defintions. In the context of agency it should be understood as a (selective) activation of agent behaviour by the environment it is situated in. This can be by sensing of the environment or receiving messages from other agents.

[5] In the context of systems theory Bunge differentiates types of black boxes by degree of participation with their environment (see Bunge [Bun79], p.253ff.).

- Autonomy – Autonomy is a controversial concept (not only) among agent researchers which has shaped two major views. Castelfranchi [Cas95], in seeking human-like equivalents of autonomy, argues that agents in fact do not necessarily incorporate autonomy and rather introduces the distinguishable types of executional autonomy and motivational autonomy in cognitive agent architectures. As such his views largely focus on what is generally understood as internal autonomy. Weigand and Dignum [WD04] accept Castelfranchi's consideration but focus on the external perspective in consideration of agent societies. They suggest to not only assume autonomy of agent entities but rather make it a requirement to assume so (which seems consequent when considering agent societies as open systems) which again implicitly emphasizes the intentional stance. From a practical point of view the degree to which an agent needs to be autonomous can vary strongly, especially when considering the granularity of the functional unit 'agent'[6]. Putting aside the external view the major differentiation of executional and motivational autonomy should be discussed as it helps to structure other related agent properties.

  Wooldridge's interpretation of autonomy (see Wooldridge [Woo97]) – to let agents make decisions about what to do with the state they encapsulate (and isolate) without direct intervention by third parties – satisfies the idea of execution autonomy. Wooldridge additionally ascribes agents own (logical) threads of control[7] over their actions and as such over their life-cycle (see Wooldridge [Woo09], p.30). In turnaround the property of *temporal continuity* – as raised by individual researchers – can be subordinated to the agent's control over its life-cycle, including a runtime of arbitrary duration. Motivational autonomy focuses on a causal relation making an agent execute an activity. Goal-directedness can be understood as a related term falling in this category.

[6] A current example where the assumption of autonomy does not hold – and which this work partially relates to – is the concept of agents in Clojure which only fulfill the criterion of reactivity, not autonomy (see [Hic10b]).

[7] Here thread is interpreted in a conceptual sense, abstaining from a 1:1 assignment of operating system threads to agents.

Although Wooldridge's actual interpretation of autonomy is pointed out above, he uses the term *proactiveness* which suggests that agents "exhibit goal-directed behaviour by *taking the initiative* in order to satisfy their design objectives."(see Wooldridge [Woo09], p.27) Taking this interpretation, proactiveness relates to both execution and motivational autonomy, in particular the latter and thus shall be captured with the concept of autonomy in this context.

Further differentiations on autonomy have been undertaken by Carabelea et al. [CBF04] who introduced several forms of autonomy (e.g. relating to the user or the environment), and Nowostawski and Purvis [NP07] who discuss the notions of relative and absolute autonomy in the context of computational systems.

- Social ability – The term social ability as listed by Franklin and Graesser implies that agents should be able to communicate using a higher-level knowledge-based (agent) communication language[8]. This way agents can interact resembling social behaviours of cooperation, coordination and negotiation. Agent researchers who suggest the necessity of advanced 'mental' capabilities like reasoning or learning – as Wooldridge (see Wooldridge [Woo97], p.3) – will accept this requirement suggested by Genesereth and Ketchpel [GK94].

  Considering a wider field of agent-based systems – more likely putting emphasis on the DIS view on agents – especially in the context of MAS (e.g. in the field of social simulation), additionally indirect communication needs to be considered legitimate in order to constitute social ability. Indirect communication involves communication via the environment using central communication mechanisms (e.g. blackboards (see Englemore and Morgan [EM88]))

[8] Agent Communication Languages will be discussed at a later point.

or situated artifacts (e.g. representing stygmergy). Although not acknowledged within the whole community[9], those mechanisms support social behaviour at least in terms of coordination and cooperation[10] – although the complexity certainly differs compared to direct 'knowledge-level' agent communication.

Some researchers go as far as to attribute social ability only to agents holding an explicit model of their communication partners (for example Moulin and Chaib-draa [MCd96]) which is additionally encouraged by high-level communication languages, such as specified by the Foundation for Intelligent Physical Agents (FIPA).

- Inferential capability – A property focusing on the internals of an agent – often directly associated with agent architectures – is inferential ability. Agents make use of information provided at initialization time or acquired during runtime in order to support decision processes[11]. It is a key feature to provide agents with the ability not only to react to predefined system states but also show seemingly intelligent behaviour and adapt their capabilities[12]. The features of inferential capability and social ability are interlinked when considering the level of communication involved (i.e. the selected language specification, the demand to represent mental components). In this context a representation 'of the self' or other agents (see property 'Character') demands for consideration.
- Character – Character (as listed by Etzioni and Weld [EW95]) or 'Personality' is suggested to be inhibited by agents. This would demand to explicitly represent a character as a believable personality including an emotional state. Although considered as a potentially relevant property, the inclusion of general-purpose concepts of emotions into agent architectures is

[9]When writing about agent communication Wooldridge does not even mention indirect communication mechanisms (see Wooldridge [Woo09], p.131ff.). Ferber serves as an example for the opposite – acknowledging the use in what he calls "purely situated MAS" [Fer99].

[10]The probably most famous examples are Ant Colony Optimization (ACO) algorithms.

[11]Typical examples include abstract task descriptions as rulesets or (meta-)information gained via interactions with the environment (including other agents).

[12]Some researchers, especially when considering user-interface agents, suggest adaptivity as a distinct agent property (see Etzioni and Weld [EW95]). In this context inferential capability, as a considerably strong agent feature, shall capture the ability of agents to adapt their behaviour and internalize new facts and apply those (learning).

comparatively young. A candidate theory to achieve similar recognition as the BDI architecture for reasoning agents is the Ortony, Clore and Collins theory (OCC theory) [OCC88] as a basis to formalize emotions as shown by Adam [Ada07]. A current example for a general-purpose approach is provided with Dastani et al. [SDM10] who include emotions into the agent deliberation cycle of the 2APL platform [Das08][13].

- Mobility – Mobility describes the ability of agents to be mobile entities changing their location during their lifetime. Especially the field of telecommunications is interesting in using *mobile code* which reduces the load on networks by *moving the code to the data* and not vice versa as in classical Remote Procedure Call (RPC) style communication (see for example White [Whi97]). Agent mobility itself is considered in a strong and weak notion: The so-called cold/weak mobility refers to the ability of connected platforms to pass agent code around without keeping its state; the hot/strong mobility also transfers the state and as such keeps the agent active during the transition phase (see Fuggetta et al. [FPV98]).

This section provides an introduction to the range of available definitions and relevant agent properties. Along with this the idea of notions of agency is introduced which themselves can be interpreted from a different standpoints (i.e. distinct definition of what makes up a strong notion (e.g. reasoning capabilities) vs. view as black box (weak notion) in contrast to white box (strong notion)). Nevertheless, the range of what *could* be part of an agent is clarified at the current stage.

2.1.2 Agent Architectures

As the definitions of agents show significant variation, the concept of notions seems more supportive in order to classify agent implementations. In this context however, the qualification of 'intelligent' was mentioned without further reflection. To clarify this boundary of agency and indicate predominant conceptual models, selected agent architectures are presented in this subsection to shed light on the range of possible implementation approaches.

[13]'A Practical Agent Programming Language' (2APL) is the simplified version of the more complex 'An Abstract Agent Programming Language' (3APL) [3APa], both developed at the University of Utrecht.

The field of agent architectures is broadly constrained by the extremes of *deliberative* and *reactive* architectures.

Deliberative agent architectures

Deliberative agent architectures put the focus on the capabilities of the individual agent and are the result of attempts to resemble practical reasoning of human beings in agents. The most-adopted concept to achieve this is the Belief-Desire-Intention (BDI) model which has been introduced into the field of philosophy by Michael Bratman [Bra87] and was adopted by the AI field respectively agent-based computing (see Rao and Georgeff [RG95]). Core elements include:

- *Beliefs* – *Beliefs* are the information an agent keeps about its environment as well as lasting conclusions it has drawn over time, commonly referred to as *informational stance*.
- *Desires* –*Desires* represent the overall motivation of an agent and eventually lead to goals which an agent tries to achieve. As such this element is subsumed as *motivational stance*.
- *Intentions* – *Intentions* are created, modified or dropped during the actual deliberation of agents and thus have a stronger practical relevance in the concept. They eventually help to identify and solve conflicts in desires and lead to goals which in turnaround can constrain future decisions.

For further reference to the core concept of the BDI model the reader is referred to Bratman [Bra87].

As of its high-level conceptual description the use of the BDI model is the blueprint for numerous implementations among which the most accepted is the Procedural Reasoning System (PRS) by Georgeff and Lansky [GL87]. The PRS system translates the abstract elements of the BDI model into five central components.

Beliefs, resulting from perceiving the environment and the actual reasoning, are stored in a *belief component/knowledge base*. A set of *goals* is held to represent the desires of an agent. Execution plans to achieve activated goals (be it by a matching set of beliefs or events) are held in a *Knowledge Area (or plan library)*. An intention structure represents a runtime component running and maintaining plans (all of which might be changed, dropped or composed). The execution of

plans eventually results in a manipulation of the environment via actuators. The *Reasoner* as central component coordinates the execution of the former components; the components do not interact directly but are mediated by the reasoner. The Reasoner repeatedly runs and controls an execution which is informally described in Listing 2.1[14].

```
WHILE there are unachieved goals DO
  Observe the environment;
  Update beliefs;
  Choose plan for execution;
  Execute and monitor plan;
END WHILE
```

Listing 2.1: Execution loop for a basic BDI Reasoner

Implementations typically consist of the aforementioned core elements and an according reasoning cycle. Actual programming languages are mostly of declarative type and inspired by Prolog. Examples include AGENT0 [Sho93] and AgentSpeak [Rao96]. Those are very appealing in this context as of their slim nature, sole concentration on facts and rules and the backtracking mechanisms for querying beliefs and matching goal/events to initiate the execution of plans consisting of several actions.

Given this description the rather individual-centric notion of such BDI-type architectures is clear. Examples of available implementations are the comparatively young Jason platform [Jasa] (using an extended version of AgentSpeak), developed by Hübner and Bordini, and the older but conceptually more sophisticated[15] 3APL platform [3APa] (with its equally named programming language), developed at the University of Utrecht.

Reactive agent architectures

In reactive architectures internal capabilities are secondary and merely reflect reactions on stimuli or perceptions of the environment. An extreme position is taken by Brooks who rejects the symbolic representation of the world as " ... the world is its own best model ... " [Bro90] and in consequence does not consider explicit

[14]This BDI reasoning loop, taken from Sterling and Taveter [ST09] represents an abstraction from the reasoning loop orginally described by Rao and Georgeff [RG95].

[15]The deliberation cycle of 3APL includes a second iteration for plan revision during execution of plans while Jason executes plans without further revision. For an overview on the individual deliberation cycles refer to [3APb] (3APL) and [BH] (Jason).

reasoning capabilities. Any 'intelligence aspect' thus emerges from an agent's reaction to its environment. An archetype for this category is the *Subsumption Architecture*, developed by Brooks [Bro86]. It consists of several layers which provide direct wiring between sensors and actuators. Each layer typically executes simple behaviours or actions (e.g. follow an object). Stacking those and associating priorities results in more complex behaviours. Brooks names those layers 'levels of competence' which is eventually reflected in the prioritization; the lower levels inhibit higher priorities (e.g. follow an object but never touch it) but the behaviour of lower levels is subsumed by higher levels (e.g. trace object = follow an object but never touch it). In consequence the number, behaviour definition and ordering of layers is fully application-dependent as of the lacking intermediate processing (as found in the BDI architecture).

Hybrid agent architectures

Hybrid approaches can be seen as a consequence of the extremes pointed out above. Those approaches neither neglect the representation of individual state in favour of a fully physical grounding nor an individual-centric approach, thus basically take elements from both extreme types. General decision to be taken for this is the definition of layers (with either reactive or deliberative function) as well as their architecture types in terms of horizontal or vertical layering. In horizontally layered architectures each layer can directly perceive the environment; vertically-layered architectures restrict this to one layer. More details on the layered architectures (one- and two-pass approaches describing the upward and downward message flow, and architectural complexity) can be found with Müller et al. [MPT95].

The often-cited example for this category, InteRRaP by Müller [Mue96], makes use of both reactive elements as well as advanced reasoning capabilities by choosing a vertically-layered two-pass architecture. InteRRaP's layers represent environment (world) interface, behaviour, plan and cooperation which can be structured into two lower layers and two higher ones. The lower (reactive) layers show immediate reaction on environmental perception, the two higher (deliberative) layers allow local and cooperative planning. Information flow between each layer is bidirectional (via *bottom-up activation* and *top-down execution*), thus all layers have

at least indirect influence on other layers while providing a reactive, individual and cooperative perspective towards goal achievement.

Most agent architectures do in fact loosely follow one or the other approach but the high-level conceptual nature – often only documented as example implementations – leave a strong engineering freedom resulting in a wide range of approaches. Nevertheless, the description of different architecture types is useful as reasoning capabilities (with deliberative architectures) represent one potential criterion to discriminate weak and strong notions of agency.

2.1.3 The Need for Dynamic Notions

Given the wide range of agent characteristics, and the strongly researcher-dependent understanding which of those actually constitute an agent, the only generally useful classification for agents are the generally accepted strong and weak notions of agency. The use of those reduces any explanation as to where the border between weak and strong is drawn. From a pragmatic point of view, and reviewing the current field of available agent platforms – the actual basis for the implementation of (multi-)agent applications[16] – this traditional split into 'weak' and 'strong' (which can be seen in the analogy to the different agent architecture types) does not suffice to capture all available solutions.

The classical bipolar association of agent platforms with regards to the supported notion of agency is only partially useful. If assuming that agent interaction by means of 'knowledge-level' agent communication distinguishes weak from strong notion, the field of existing agent platforms cannot be covered comprehensively. Popular platforms[17] accounting for the strong notion include JADE [BCG07], 3APL [3APa] and Jason [BWH07]. Opposing examples for platforms rather supporting the weak notion of agency, not assuming high-level communication include MadKit [GFM00] and Cougaar [HTW04]. However, considering platforms such as the Otago Agent Platform (OPAL) [NBPC01] which has an integrated multi-level model of agency or Jadex [Jadb] which has a non-integrated dual approach to

[16] At this point this should satisfy the necessary understanding of an agent platform.

[17] Overviews on most of the mentioned platforms are provided in [FNP10] and [Fra09].

model agents[18] in fact satisfy both categories. In order not only to allocate platforms such as OPAL and Jadex more specifically but also the provide a communication vehicle in different contexts, a further dimension is suggested at this point, in order to augment the yet rather *static concept of weak and strong notions*.

Along with the static notion indicating a discrete agent classification (i.e. strong or weak), platforms can be characterized by their *dynamic notion*, describing the flexibility the platform inhibits with regards to the supported agent notion. *Narrow* notions describe rather strict limitations and narrow interpretation of what agents are. Jason, for example, is strictly confined to the use of coarse-grained 'intelligent' agents which communicate via KQML and are consistently implemented in AgentSpeak; it shows a *strong narrow* notion. MadKit in contrast supports a considerably weak notion of agency, hardly assuming anything about agents but also providing an intentionally small-scale infrastructure which does not support coarse-grained agent concepts out of the box; its notion is *weak narrow*. Platforms with a *wide* understanding include OPAL which is built on the notion of so-called micro-agents which weaken some of the core properties (e.g. standard-orientation, means of communication) in favour of flexibility and (performance) efficiency.[19] Those are complemented by OPAL agents which are standard-compliant (with FIPA standards) and cater for the strong notion of agency. More precisely, OPAL can account for a *weak wide* notion of agency as OPAL agents eventually are an extension of micro-agents (i.e. the core of each OPAL agent is at least one micro-agent). Contrasting this a platform supporting a *strong wide* notion is Jadex. The Jadex core is centered around intelligent agents (as of its origin as a BDI implementation extending the JADE platform). However, it alternatively offers a weak notion of agency, also called micro-agents[20], and as such equally covers the full field of agent notions.

The additional dynamic dimension of agency not only covers the classification of existing agent platforms but is also applicable in alternative contexts. On a

[18]Here it should be mentioned that the two levels of agents have only been introduced in the recent release of the platform. Before that Jadex had been solely committed to BDI agents (see [PBL03]).

[19]Further explanation will be given at a later stage.

[20]The motivation for their use is similar as in OPAL, however, the Jadex understanding and level of elaboration of micro-agents differs.

higher level this is the description of trends (of changing agent perception) in research fields. The increasing distance from standard-orientation (such as FIPA (see subsection 2.2.3)) can be seen as such a trend; research fields with differing understanding of agency include the AOSE community – with a historically stronger concentration on heavy agent notions – in contrast to the ABSS community which emphasized a significantly weaker but increasingly heavier notion of agency (originated from (pre-agent era) Microsimulation moving towards the introduction of 'intelligence' characteristics[21]). On a lower level – and covering future research aspects of both agents in general and platforms in particular – it might not suffice to strictly confine agent implementations to either one category. The dynamic notion concept allows a continuous understanding of agency purely grounded on the intentional stance as a minimum – enforcing agents to be seen as rational entities and further assuming that they incorporate the necessary autonomy (as suggested by Weigand and Dignum [WD04]). How and if this is achieved should not be of concern. Although the adaptivity of agents has been suggested (see subsection 2.1.1), no agent platform implementation has yet realized this in an actually dynamic manner. New, however, is the suggestion that agents should be able to rightsize their capabilities as runtime, i.e. scale up/down functionality (i.e. actually give up capabilities) in a demand-oriented manner to achieve system optimizations at runtime (e.g. to be cooperative towards other agents by giving up (limited) system resources or yield towards an optimized system performance). Agents of *wide notion* should thus be capable of having a small initial feature set – perhaps purely reactive behaviour – but eventually start to act proactively and initiate conversations up to a social level in which they keep a mental model of their interaction partner. Agents with an initially 'heavy' feature set, eventually give up reasoning capabilities in case of role changes (as they might not need it anymore) and act simply responsive, e.g. driven by the goal to minimize use of system resources on mobile devices. As such the term 'dynamic' in this context does not relate to either one extreme but describes the adaptivity of an agent's feature set.

[21] This will be introduced in subsection 3.2.1.

A *narrow notion* in contrast describes agents which are strictly confined to either one static notion – which captures the traditional understanding of agent notions. The actual dimension aspect is realized by the combination of dynamic notions with static ones. The implication for narrow notions is clear – it emphasizes the strictness of its static characteristic; an agent of narrow strong notion will not weaken its internal capabilities at runtime, an agent following a narrow weak notion will equally remain an agent of weak notion at runtime. The opposing wide notion suggests an agent which initially incorporates reasoning capabilities and potentially gives those up at runtime as part of the system's or its own objective. Clarifying the dynamic notions on the lowest level, respectively in the agent imple-

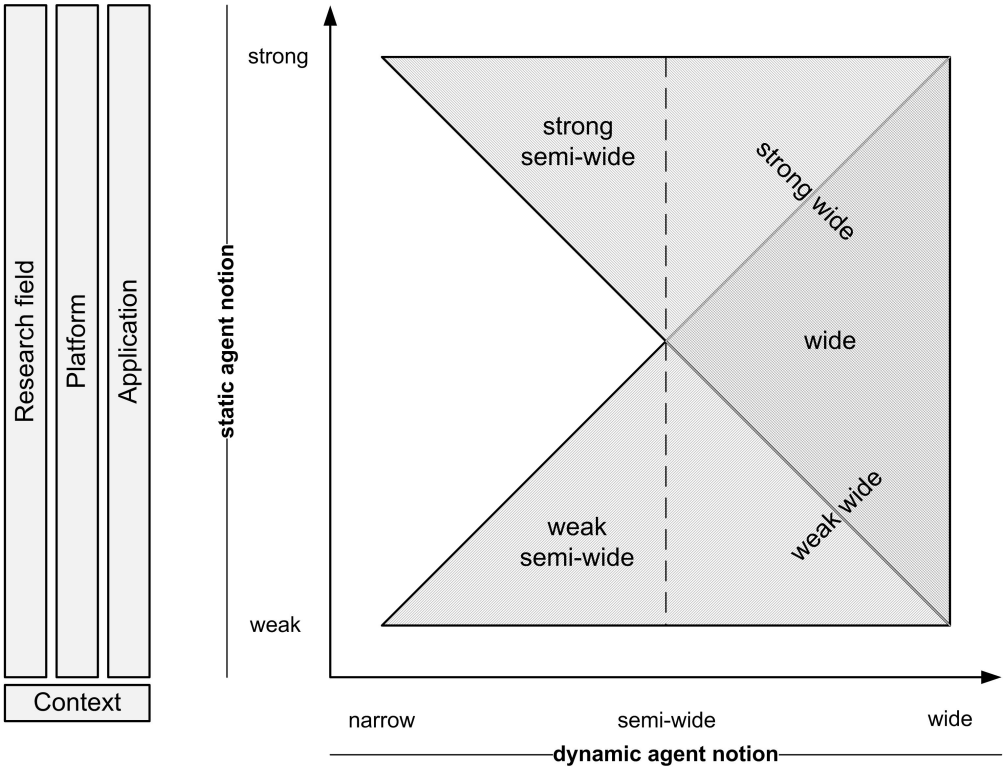


FIGURE 2.1: Dimensions of Agent Notions

mentation context, it needs to be said that dynamic notions describe potentials. The non-deterministic nature of the agent-based systems (in the DIS understanding) does not allow a reliable prediction if a capability change will actually take place. As such the dynamic notions encapsulate the static notions (which describe the initial agent notion) at runtime. In consequence the actually dynamism of a system can only be determined by a runtime analysis of an agent application

implementation.

Figure 2.1 shows how the introduction of the additional dimension extends the range of potential agent characterisations. Those are augmented with an according context. The interpretation is done in either one of those contexts. This could be the description of the agent concept in a specific application, a means to characterize platforms or to describe trends in entire research fields. Additional to this Table 2.1 interpretes the dynamic notion concept for the context of Multi-agent platforms.

Tying together the three mentioned contexts, application, platform and research

Supported Agent notion	**Description of Agent Modelling Potential in Platform**
weak narrow	support for weak agency; only very simplistic agents can be implemented (e.g. ants)
weak semi-wide	support for several agent notions ranging from a weak understanding towards stronger notions (e.g. advanced social behaviour/communication abilities) but *not* to the extent of a strong agent notion (e.g. BDI-level reasoning engines, standard-compliance)
weak wide	support for weak agent notion up to the extent of a strong agent notion; weak agent notion as dominant/default approach
strong narrow	support for strong notions of agency; light-weight agents cannot be implemented
strong semi-wide	support for strong agent understanding per default but allows modelling of weaker notions (e.g. no BDI-level reasoning), however, *not* to the extent of a purely weak agent notion
strong wide	support for strong notions of agency per default but allows the modelling of weaker agent notions up to a level of simplistic (e.g. ant-like) agents
wide	support for the full range of agent notions with equal preference

TABLE 2.1: Overview of combinations of Notion Dimensions for Platforms

field, it should be beared in mind that those do not coexist in an isolated manner. As agent platforms constitute the basis for any agent implementation, i.e. define a potential agent feature set, they constrain the achievable dynamics of an agent application; research fields constrain what is considered an agent platform. However, particularly for the context of agent applications – the lowest level context – it should be mentioned that the dynamics on this level are yet to be realized;

current agent platforms do not generally cover the flexibility aspect suggested here yet.

In the context of this work the dynamic notions of agency shall be used to characterise the agent understanding of specific platforms and within research fields – rather than prescribing a distinct agent understanding for the entire thesis.

2.2 Multi-Agent Systems

The introduction of the broad agent concept provides the core ingredient of a Multi-Agent System (MAS). However, even popular multi-agent researchers in fact concentrate on the single agent definition and fall short on explicitly dealing with the MAS concept but rather simply suggest the multiplicity of interacting agents[22]. As the super-fields of information systems and information science in general are born as an interdisciplinary application of systems modelling it seems indicated to not only provide a sound understanding of what constitutes a Multi-Agent System but also to retrace ideas from the Systems Theory in general.

2.2.1 System-theoretical Foundations

Although the term *system* provides proof for its antique descent by its root in the Greek *systēma* ("... whole compounded of several parts or members ..." [Sys]), the foundation for a cross-disciplinary generalized system understanding was laid in the 20th Century by, among others, von Bertalanffy in his General Systems Theory (GST) [vB68]. Main achievement of this work was the introduction of a general system theory yielding at bridging the "... largely artificial barriers between disciplines." [Bun79], and as such to overcome the predominant reductionist paradigm advocating the isolated examination within scientific disciplines. Von Bertalanffy specifically aimed at "... developing unifying principles running vertically through the universe of the individual sciences ..." [vB68], explicitly including the consideration of the non-physical sciences.

One approach to sharpen this theory respectively conceptually unify it[23] is done by Bunge [Bun79].

[22] Wooldridge: "Multiagent systems are systems composed of multiple interacting computing elements, known as agents." [Woo09]

[23] Bunge explicitly critizes that the GST in fact consists of numerous theories merely unified by a philosophical framework (see Bunge [Bun77]).

Bunge uses the term of *systemics* and differentiates between a *cognitive* or theoretical motivation – identifying the commonalities between systems of different disciplines – and a practical one, the latter bearing the ability to describe the variety of different real (and artificial) systems such as nations, software and factories with all their peculiarities (see Bunge [Bun79]). In the following this framework will be retraced which, apart from illustrating examples, is based on Bunge [Bun79] which should be consulted for a comprehensive insight.

Although offering a theory (and a methodology as to be shown later) to describe wholes, systemics reject both the extremes of holism and atomism. While holism emphasizes the integrity of a system in favour of its components and their interactions, atomism basically describes the opposing bottom-up approach, assuming that the study of the individual components suffices to understand a system's whole. Although certain characteristics (like considering the whole '... more than the sum of its parts ..') are related, Bunge remarks that consequent holism hinders research by its lack of interest in (potentially confusing but relevant) details. While atomism stimulates the investigation of the components (and as such research) its (typically reductionistic) approach is not capable to sufficiently capture higher-level system characteristics such as emergence (see Bunge [Bun79], p.41).

Bunge's system concept consists of a terminological framework (along with its formal equivalents) which not only describes the static structure but puts emphasis on dynamic characteristics of systems, which constrasts them from the ontological view advocated by holism.

The general concept of a system σ consists of its composition C, its environment E and its structure S, thus is expressed as $\sigma = <C, E, S>$.

The composition and the environment are mutually disjoint subsets of a given set of conceptual items I, i.e. $C(\sigma) = (x \in I \mid x \notin E(\sigma))$. Its structure is the non-empty set of relations R on the union of both C and E. The latter characteristic (R) is key to satisfy the requirements constituting a system. The relations between system components – of whatever specific quality – differentiate it from the concept of an aggregate which is merely a collection of items (or 'set' as Bunge names it). Although this minimal definition provides a basic coverage of the system idea, a

term to be considered in the context of any kind of system is whether it is an open system, or a closed one. Bunge suggests the universe as the only natural system known to be closed at all times. A system is considered closed with respect to one of its properties if the latter has no relation to a property of the system's environment. In consequence a system is closed if its environment equals nil.

A concrete system extends the general concept by refining the conceptual items into things, or *concrete entities*, and introduces the notion of levels and subsystems. In consequence a composition in this context refers to the set of system atoms *at a given level* rather than its set of heterogeneous parts. An illustrating example from the field of tournaments in team sports such as in soccer is the concept of teams as such. Team systems are made up by team members (which represent the atomic elements of teams). Those are in relation to each other – be it closer (as of similar position and frequent interaction (e.g. striker and mid-field player)) or less close (e.g. striker and goal keeper). Although the dynamics of the game are enabled by the players' 'running capabilities', the players' legs do not generally directly interact with other players' legs[24] or the team as a whole but belong to the atomic concept player. The level of the system 'team' is a composition of the socially connectable seemingly atomic element player. The concept player in itself might be decomposable into another system of different *natural kinds* (e.g. organs) but those do not directly interact in the context of the system team (which has players as its natural kinds). Important for the term 'level' is the fact that all elements x of a succeeding level[25] L_b are composed from things y of the preceding level L_a, i.e. $L_a < L_b = (\forall x)[x \in L_b \Rightarrow (\exists y)(y \in L_a \ \& \ y \in C(x))]$.

Relations in concrete systems can eventually result in an effect on either one or both of the related parties. The latter specialization is coined *connection* or *bondage* and changes the *history* of either one (by acting) or both (interacting).[26] The set of relations in concrete systems is thus understood as the union of the *bonding* and *non-bonding relations*. Those different qualities of relations allow the

[24] Cases of direct 'leg contact' shall be ignored at this point.

[25] Bunge avoids the term 'hierarchy' as it etymologically describes "a set of sacred components ordered by a power or domination relation" (Bunge [Bun79], p.12) which is inadequate to describe any system relation.

[26] For the full terminological differentiation and further discussion of details refer to Bunge [Bun79].

use of the term *organisation* for the system structure.

The environment in a concrete system is defined by all things of the system (i.e. its natural kinds of its level) which are not part of the composition of the system and are in an active or passive connection to each other.

The introduction of a temporal dimension does not only enable the representation of temporal states in a system but also improves the understanding of what makes systems different from aggregates. A temporal perspective provides means to represent the impacts of (inter)acting things affecting the history of either one and in consequence the system as a whole. This provides a more subtle explanation of the added value of the system concept than the "fuzzy slogan of holistic metaphysics ... [']The whole is greater than the sum of its parts.[']" [Bun79].

The preceding introduction of a system's constituents allows the formulation of the minimal definition[27] of a concrete system σ on the level P at the time t as $s_P(\sigma, t) = <C_P(\sigma, t), E_P(\sigma, t), S_P(\sigma, t)>$.

Along with the definition of the necessary elements of a system the order of definition components indicates methodological steps in systems analysis, giving a concrete suggestion of how to explain systems in general: As such first the components need to be defined. Following this the environment is to be identified. Links between components as well as components and the environment then describe the structure respectively organisation of the system of concern, completing its specification.

Yet the definition of a system leaves out general system characteristics which are briefly retraced to provide a sufficiently complete picture – and to bring us closer to the concepts which are all but foreign in the field agent-based systems. Basic behavioural processes of systems concern the assembly and emergence. *Assembly* is the process of creating bondages within an unchanged composition and is further specialized in *self-assembly* if an aggregate pursues this transformation to a system on its own, thus in a natural manner. The strongest case of assembly is *self-organisation* which describes the process in which a system creates subsystems on its own behalf. While assembly refers to the structural aspects of a system, a

[27] For the full definition of a concrete system, Bunge insists to have full knowledge about the history of each component as well as the "laws of the system" [Bun79].

process concerning the properties of the composed (system) elements is *emergence*. As such emergent properties are the difference of an element's properties occurring within a time interval, characterized as either a gain or a loss – focused on the individual element in a system's composition.[28]

Measures concerning systems are relevant as a system's elements, although establishing a whole, do not give up their individual existence. Measures such as the level of *integration* or *cohesion* (which can potentially be measured continuously) determine the strengths of the existing links between the elements. Measures of the latter enable to describe the critical size of a system – the maximum degree of integration of a system. Observation of systems when considering their composition of subsystems motivate Bunge to postulate that ".. [the] more cohesive each subsystem the less cohesive the total system." [Bun79] As such not only the composition of a system concerns multiple levels, the integration and stability of the total system does as well. Processes of disintegration of a system are characterized as *structural breakdown*, while failure of *coordination* would result in a *functional breakdown* which does not necessary result in a structural breakdown. The fact that a system 'does not work' at a given time, is not equivalent to the disintegration of its structure. An example is a human being's death which represents a functional breakdown – not its immediate structural breakdown.

Summing up, this subsection has introduced the term 'system' including the two general kinds of systems (conceptual and concrete systems), discussed the structural elements (composition, environment and structure), several degrees of behaviours or processes (assembly and emergence) and general often measurable system characteristics (integration, cohesion and coordination). This framework taken from Bunge also supplies a basic but valuable prescription on steps to pursue in order to evaluate or model systems in general.

2.2.2 Multi-Agent Systems

As both the 'agent' and 'system' concept are sufficiently discussed, we have the foundation to construct the full 'Multi-Agent System' (MAS) concept – and know

[28] Processes like selection and evolution are not further described in this context but receive attention by Bunge [Bun79].

that the stakes for this are higher than just being an aggregate of agents as suggested in the beginning of this section.

Ferber [Fer99] distinguishes several elements which qualify a multi-agent system. As such a MAS consists of an *environment* with a volume. This environment is populated with *objects*[29] which at any given point in time have an unambigious association with a distinct position in that system.

A MAS also comprises of *relations* (R) between objects; *agents* (A) are interpreted as active specializations of objects and are as such included in this context.

Ferber additionally suggests the assembly of *operations* (Op) by which agents can act upon the environment, namely ".. perceive, produce, consume, transform and manipulate objects" [Fer99]. At this point it can be suggested that the dependence of the operations (Op) on agents (i.e. operations are executed by agents but not every agent needs to be able to execute all of them) makes an association of operations to agent entities useful (Op_A) and avoids the risk to perceive those operations as 'free' (in terms of 'unassociatedness').

Additional to this *operators* (in order to clearly distinguish this from the aforementioned, *feedback* might be an alternative term), here abbreviated as Fb, represent the *application or executions* of the aforementioned operations *in* the world and the according *reactions of* the world. In Ferber's concept those are described as "laws of the universe" [Fer99]. Applying Ferber's operators to Bunge's system understanding, the presence of those operators (and the obviously existing direct or indirect relations between environment and agents in the system) discriminate whether a system is open or closed.

For these operators (Fb) a clarification is suggested at this point. As multi-agent systems are artificial systems (whether or not they try resemble natural ones), they allow the definition of a comparatively controlled environment[30] respectively can represent a closed system[31]. However, applications built in top of multi-agent

[29] Objects allow at least some (if not all) of the following core functions: to be created, perceived, modified or destroyed. (The analogy from persistent storage systems are the CRUD operations (Create, Read, Update, Delete).)

[30] Concerns of determinism are raised at a later point.

[31] Again, this needs to be seen with some conceptual abstraction. MAS infrastructure is certainly all but independent from external events (such as operating system failure, power loss), thus can never be a closed system in the strict sense.

systems are always open systems with regards to the underlying MAS as their infrastructure, 'their' universe. The 'laws of the universe' (Fb) are thus consistently affecting the application, even when modelling MAS applications as closed systems, such as typically the case for experimental settings. Therefore describing the openness of a MAS is a concern of the environment understanding involved. MAS implicitly involve at least two types of environments, an *application environment*, which is modelled (or eventually not modelled!) by the application developer, and an *infrastructural environment* (agent communication, scheduling) which is inevitably provided with the runtime platform – be it integrated with the application or separated from it (see subsection 3.2.3 for a further discussion on MAS for simulations). An empty (application) environment alone does not result in the absence of those general laws; the general laws of the infrastructural environment continue to apply to agents across multiple levels of a system. An example is the consideration of an application with agents acting in an empty environment – coined as *purely communicating MAS* [Fer99]. In this case the infrastructural environment is in fact identical with the application environment.

The opposing extreme in which agents solely interact via the (then typically modelled application) environment by *signalling* is what Ferber [Fer99] understands as *purely situated MAS*.[32]

Given the core components of MAS, the analogy to an abstract system (see subsection 2.2.1) can be drawn. An abstract MAS could thus be expressed as $\sigma = (<C(O), E, R> \;\&\; \forall x\ (x \in A \mid x \in O) \;\&\; (\exists y\ (y \in A) \;\&\; \exists z\ (z \in A) \;\&\; y \neq z))$:

A MAS consists of a composition of objects O representing the components of an abstract system, an environment E and the structure defined by the relations between objects. Agents A represent a specialization of objects, i.e. all agents are objects. At least two different agents are necessary to consider it a multi-agent system – as opposed to a multi-object system.

Introducing the notion of levels – as considered in the context of systems anyway – the overall structure of MAS as multi-level systems is only consequent – especially when using MAS to model complex software systems which is the key purpose of

[32] In this context the notions of communicating vs. signalling are equal to the concepts of direct vs. indirect communication.

AOSE (see section 3.1). Again, in analogy, Bunge's definition of system levels, as provided in subsection 2.2.1, can be directly applied in the context of MAS and is repeated here for the sake of convenience: ($L_a < L_b = (\forall x)[x \in L_b \Rightarrow (\exists y)(y \in L_a \ \& \ y \in C(x))]$)

Considering L_b to represent a higher level – or macro-level from a sociological point of view – its entities (agents respectively objects) y are decomposed into their counterparts x on the lower level. Important aspect is here that entities from the higher level do not directly interact with entities on the lower level but in a mediated manner (e.g. via infrastructure or representative unit).

In the consequence modelling of MAS as multiple levels of organisations (thus consisting of agents which are themselves decomposed into MAS organisations) is only but natural. The ultimately emerging recursive structure will eventually bottom out on an atomic agent level[33]. First concepts in the area of organisation-centric MAS – introduced about a decade ago – are the Agent-Group-Role model [FG98] which is the meta-model of agent organisations in MadKit [GFM00], and the KEA micro-agent architecture [NPC01], building the core of OPAL. A full introduction of both platforms and meta-models will be avoided at this point; instead their consistency with regards to Bunge's understanding of system levels is compared.

MadKit provides a considerable – though as far as literature is concerned unintentionally[34] – pure implementation of Bunge's system level understanding, especially considering the communication across system level boundaries. MadKit uses dedicated roles taking over the communication with external agents. A drawback of this conceptually clean approach is the performance penalty involved. OPAL's micro-agents in contrast, take a lenient approach with regards to communication. Any micro-agent can potentially address any other micro-agent.

Another aspect concerns the organisation: Micro-agents have an implicit handling of groups while MadKit demands for explicit specification by the developer. Micro-agents are thus consistently organised in groups; in MadKit this cannot be

[33]Retracing this image it should be emphasized that it does not necessarily imply a top-down approach on modelling MAS.

[34]No indication for this is given in the literature surrounding MadKit (such as [GFM00] or Ferber himself [Fer99]). In fact Ferber, similar to Bunge's criticism of the GST, considered systemics unsuitable as a MAS blueprint (see [Fer99], p.53f.) but did not take Bunge's system understanding in account.

ensured respectively needs to be handled explicitly. Apart from this micro-agent groups are inherently organised in levels, allowing agents to maintain subsystems respectively sub-agents. Levels are handled in a hierarchical manner (respectively as levels of systems) with a platform-controlled agent at the top which allows a consistent view on the overall agent organisation across the entire platform.

The brief comparison shows both specific modelling advantages for either model. MadKit provides a compliant communication mechanism while micro-agents ensure the semi-automatic consistent embedding in an overall organisational structure. Further information on MadKit is provided in its documentation [GFM00], the micro-agent concept will be target of discussion in chapter 5.

Levels in MAS are not only interesting from a system-theoretic perspective but have a strong sociological link. The concept of organisational levels in the field of sociology (documented by Gurvitch [Gur63]) identifies three levels which also find acceptance among MAS researchers (such as Ferber [Fer99]). At the lowest *micro-social level* the interactions between few agents, often of equal capabilities, are studied. On the next higher level, the *group level*, the development of group structures of agents, is studied, predominantly focusing on the development of organisations. The level of *populations* is concerned with a more general view on a system's emergence and its structural development. While the micro-social level reflects the domain of micro-sociology, the latter two belong to macro-sociology[35] which focuses on the clusters arising from the micro-social level. The macro-social level concentrates on the observation of emerging structures and levels of equilibria. Although the different sociological levels are not of particular concern for the applied part of this work, they are essential in the context of social simulation and the explanation of emergence.[36]

The properties motivate to relate particular organisation levels on equivalent application fields in the context of MAS. Table 2.2 structures the organisational levels and augments those with characteristics for according MAS applications (and is an extension to the set of characteristics provided by Ferber (see Ferber [Fer99],

[35]The differentiation of micro- and macro-sociology is described by Gurvitch in [Gur64].

[36]Here to mention is the downward and upward causation effects of the Coleman boat (see Coleman [Col90] or, on a higher level, Troitzsch [Tro09b]).

p.13f.)).

The suggested application patterns can certainly not always be generalized as the

Organisation level	Micro-social	Group	Population
Process Objectives	Coordination	Organisation	Evolution
Importance of agent lifecycle	low	low - middle	high
Number of agents	few	many	massive
Agent notion (platform context)	wide	wide weak	narrow weak
(Originating) Research field	DAI	MAS	Massive MAS

TABLE 2.2: Characteristics of organisation levels

actual characteristic of an application is task-dependent but allow the identification of characteristics relevant for the related MAS research fields.

The micro-social level deals with a limited number of agents whose coordination is the core objectives of the surrounding system. The relevance of full lifecycles is often limited. As coordination between individual and mostly 'intelligent' agents (e.g. playing a game) is of relevance, the lifetime of agents is often equal to system runtime. Originating research field is the area of Distributed Artificial Intelligence (DAI). In the context of multi-level MAS, i.e. MAS consisting of subsystems which are MAS themselves, the micro-social level is relevant on various levels and does not necessarily emphasize individual 'intelligent' activities. Nevertheless coordination remains a core objective. Application areas include auctions respectively game-theoretic investigations.

The group level deals with a larger number of agents eventually engaging in a process of organisation and allows the observation of (emerging) group behaviour. The lifecycle importance is rather low, the agent notion is typically weaker, autonomy of agents more limited and the use of sophisticated internal representation hardly found. As such the system is interaction-centric rather than agent-centric, emphasizing interaction in contrast to intelligence. Simulation scenarios such as the one presented in the context of this work (see chapter 6) often fall in this category.

The organisational level of populations refers to a category of systems representing strong dynamics within agent populations, modelling full lifecycles, i.e. birth and death representation of organisms (e.g. micro-biology), as found in the areas

of artificial life and evolutionary computing which changes the perspective from individuals or groups to generations. The number of agents found in systems of this kind can be considered massive (from several hundreds upwards); the agent concept is typical very weak respectively loses most of core agent qualities eventually being nothing more than a method call - which is rather an optimization result enforced by constrained computing power than the need for simplified individuals. As such the use of the latter organisation type is (and was) mostly constrained by computational resources; it will gain stronger relevance with the ongoing increase of computing power as all levels will presumably 'scale up', be it the more complex internal representations on group levels or increasing numbers of agents on the micro-social level. Along with this, or alternatively, the choice of development languages can switch from an efficiency/system orientation (e.g. C, C++) to a convenience/application orientation (e.g. Java, JVM-based languages, interpreted languages).

2.2.3 Standard Specifications for MAS

Although not of core relevance for this work but often mentioned without further introduction, the standards for MAS should not be ignored. They are historically important, still in use (e.g. by OPAL and JADE) and represent one extreme in the trade-off between openness and performance.

The end of the 1980's until the beginning of the first decade of this century mark a period of active research with regards to the issue of interoperability respectively openness in agent-based systems. A standard focusing on the issue of agent mobility is the Mobile Agent System Interoperability Facility (MASIF) [MBB+98], published by the Object Management Group (OMG). Idea is the agent platform-independent provision of mobility by integrating the concept of agency, place and region (group of agencies that belong to a single authority) along with platform lookup functionality and a defined transition process. The actual means of communication are not concerned; the standard is seen as complementary to the CORBA standards which are supposed to specify communication as well as security. The

mobility aspect is primarily driven by the telecommunications industry; the standard did not find general acceptance in mainstream platforms[37]. A platform known as a reference for its use is Grasshopper [BBCM99][38].

Other (more popular) standards are communication-centric, such as the Knowledge Query and Manipulation Language (KQML) [FFMM94] and several specifications provided by the Foundation for Intelligent Physical Agents (FIPA) [FIPf]. Especially the latter represents the most comprehensive specification for the concept of *Agent Communication Languages* (ACL).

Both standards base on the speech act theory introduced by Austin [Aus62] and popularized by Searle [Sea69], which essentially switches the perspective on communication analysis from earlier approaches of conceptual mapping of words (e.g. the 'early' Wittgenstein [Wit21]) to a high-level semantic one, interpreting communication as action. Given this, certain mental states (i.e. beliefs, expectations) of sender and recipient can be assumed. Searle [Sea76] later classified five possible types of speech acts. Those are listed below, along with selected *performatives*[39] reflecting those types in ACLs (where applicable):

- *Representatives* bind the speaker to the truth of a proposition and reflect the idea to *inform* a recipient about a certain (believed) fact or condition.
- *Directives* are attempts to make the recipient perform an action; a *request* is a commonly used example for this.
- *Commissives* classify acts which bind the speaker to a certain action, such as a *promise*.
- *Expressives* verbalize attitudes and emotions of the speaker, such as gratitude (e.g. *to thank*) or personal judgement of some kind.
- *Declarations* are the final group and include actions which have the power to affect " ... the institutional state of affairs."([Woo09], p.134) An example is to *declare* a couple as husband and wife.

KQML is the first attempt to use performative-based communication for interaction between different knowledge-based systems. It makes use of a high-level

[37]Saber and Ferber discuss the practical limitations for the use of the MASIF standard [SF].
[38]The platform itself, however, is not provided anymore.
[39]The performatives are emphasized for each according speech act (e.g. *inform*).

communication and is independent from a knowledge domain which makes its use appealing in the context of agent-based systems. Its syntax is LISP-like – using balanced-parentheses lists. The standard considers primitives for network management and administration (i.e. network level functionality). The embedded content language for actual expressions can be chosen by the developer. Examples are Prolog and the Knowledge Interchange format (KIF) [Gen].

The disadvantage of KQML is its loose semantics. Its use in applications thus led to numerous incompatible dialects.

The FIPA ACL [FIPb], as part of a considerably more comprehensive set of specifications, evolved from the Arcol standard developed by France Telecom, and targets towards solving the deficiencies of KQML. As a reaction the specification of the FIPA ACL concentrates on semantics, strongly reflecting the implications of speech acts[40] on both recipient and sender side (such as the binding effects as mentioned above), and in conjunction a content language (FIPA SL [FIPe]) which allows the expression of beliefs, uncertain beliefs, desires and action along with a mapping to the semantics of the ACL. The ACL syntax itself is equivalent to the one of KQML. As a result, the FIPA specifications constitute a domain-independent and less ambigious 'knowledge-based communication' by introducing strong semantic constraints.

The downside of the FIPA ACL in turnaround is two-fold. Firstly, the correct implementation across different platforms can hardly be tested as only basic interoperability tests had been conducted at given times[41] – rather to ensure the unambiguity of FIPA standards than ensuring implementation conformance. FIPA itself has left this issue largely to the "market-place" [IEE], thus conformance can hardly be assumed. AgentCities, a community-driven project targeting interoperability testing as well as service provision on a global scale ceased to exist as well. Today platforms implementing the FIPA standards are very limited, have often limited support for the full FIPA specification set and are typically not tested

[40] Its specification for according *Communicative acts* can be found under [FIP02].

[41] The first interoperability test was held in Seoul in January 1999, the second one in London in April 2001. Apart from the JADE platform [JAD09] none of the other platforms (FIPA-OS [FIPd], ZEUS [ZEU], April Agent Toolkit (which had been integrated into another framework) [Apr]) is actively maintained.

on this[42]. As such the only reference implementation available can be considered JADE [JADa].[43]

The second problem apart from conformance testing is the underlying intelligent agent assumption: FIPA agents, although not strictly specified, need to be able to represent the concepts of beliefs, desires, intentions and goals. This implicitly enforces a rather heavy-weight agent architecture, confining not only developers to this model but also limiting the social sphere of FIPA-agents to agents of 'their kind'. As FIPA communication still allows ambigious interpretation and relies on interaction protocols[44] as support construct it is target to criticism. Singh [Sin98] argues that the assumption of agent internals (i.e. BDI-like capabilities) shows the insufficiency of ACLs to really represent open communication on a knowledge level.

Concluding this short presentation of relevant standards for the MAS development the FIPA Agent Management Specification is mentioned as it provides a conceptual blueprint for agent platforms in general, although actual implementations often do not follow the specifications.[45] According to this specification an " .. agent platform (AP) provides the physical infrastructure in which agents can be deployed. The AP consists of the machine(s), operating system, agent support software, .. agent management components ... and agents." [FIP04] In this model agents are autonomous communicating entities which have both an owner as well as an *Agent Identifier* (AID) which is globally unique. The *Agent Management System* (AMS) is concerned with the maintenance of an agent directory on the according platform (and, among further functions, assigns the AID). Another component is the *Directory Facilitator* (DF) which provides a yellow-page service to look up agents based on their capabilities. Multiple DFs can coexist on a platform and potentially undergo federation relationships. AMS and DF are, along with

[42]In fact nearly all platforms known to implement FIPA specifications have ceased to exist (see FIPA website [FIP03] and Wikipedia list [FIPa]).

[43]The Otago Agent Platform (OPAL) mentioned here had been tested on compliance with the Agentcities project but was not involved in the original interoperability tests as of its relatively younger age.

[44]For an elaboration on Interaction Protocols respectively Conversation Policies see Greaves et al. [GHB00].

[45]Here the analogy to the often-referred but never widely adopted Open Systems Interconnections (OSI) Model, specified by the International Organization for Standardisation (ISO), could be drawn.

the 'application agents' (i.e. all agents developed by the application developer), backed by a common *Message Transport System* (MTS), which serves as default communication mechanism as specified in [FIPc].[46]

Figure 2.2 depicts the aforementioned components and their relations as provided with the original specification [FIP04].

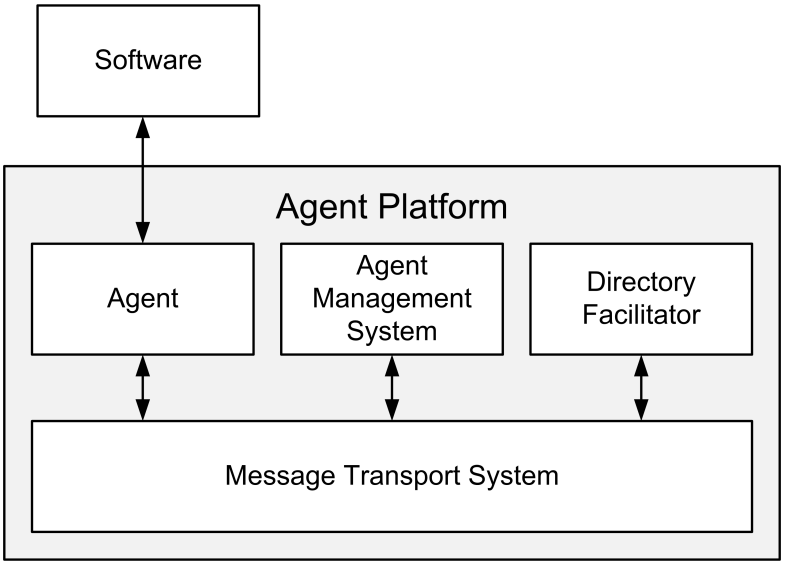


FIGURE 2.2: FIPA Agent Management Reference Model

[46]Its implementation-independence is expressed by the specifications for transport protocols which include IIOP, HTTP and SMTP.

Chapter 3

Research fields in Agent-based Computing

Given the background on core concepts in the context of agent-based systems, the context of their application will be discussed. Although implicitly shown before for different levels of observations in MAS a better insight into two disciplines which see and apply MAS from a different perspective should be given. One of those is the field of Agent-Oriented Software Engineering which uses MAS as modelling paradigm for the development of information systems, the other Agent-Based Social Simulation which is driven by an interdisciplinary community and sees MAS as a particularly suitable (as of its sociological relation (see subsection 2.2.2)) blueprint in order to effectively model and understand real-life problems. In the following each of those is introduced in a dedicated section. Their differences and the implications for according multi-agent platforms are discussed in subsection 3.2.3.

3.1 Agent-Oriented Software Engineering

3.1.1 History and Principles of AOSE

The advent of the idea of Agent-Oriented Programming (AOP) introduced by Shoham [Sho93] is based on understanding multi-agent systems as intentional systems incorporating notions of belief, desires, intentions and goals and yields towards programming in terms of mental states. Shoham himself sees AOP as a

specialization of Object-oriented Programming (OOP) and interpretes the state of those specialized objects (agents) as 'mental state' whose use is constrained by defined components such as beliefs, capabilities and further elements. Interaction between agents is restricted to specified message types (identified by performatives), based on ideas from the speech act theory including the representation of presuppositions and effects for the related communicative acts (see Shoham [Sho97]). In analogy to the mental level of communication Wooldridge sees AOP as "post-declarative programming" ([Woo97], p.5), following the idea to declare some goal to be achieved by the system while abstracting from the actual control mechanisms implemented in the language (as done in Prolog). The transition he considers is the replacement of the language-internal control mechanism with an implementation of a rational agency model such as the BDI model (as introduced into the field by Rao and Georgeff [RG95]) which would then act based on the mental components.

The first language following the AOP paradigm is AGENT0 [Sho93], developed by Shoham himself. Along with the BDI model as a popular model for rational agents, AgentSpeak [Rao96][1] has gained wide recognition, lately especially for its use in the Jason platform [BHW07]. A view on agents as intentional entities and the development of an according programming paradigm allowed the embedding of those elements into a methodological context which Wooldridge and Jennings named *Agent-Oriented Software Engineering* (AOSE). Objective of AOSE is to narrow the gap between the (among itself all but unified) research community and software engineers in order to enrich the toolbox of everyday software development with agent-based concepts.

The development of complex systems is a key area in which agent-oriented software engineering promises to show significant advantages against other approaches such as the classical OO approach – especially considering a continuous system extension. As the complexity is not unintended, but more or less explicitly and successfully engineered, certain rules can be derived for those artificial systems, as done by Simon [Sim96]: Complex systems often involve hierarchies whose strength

[1] Its original name was AgentSpeak(L) but is widely referred as AgentSpeak today, including its various dialects.

and structures vary depending on the subsystems involved. This concerns the temporal dimension (change over time) as well as the concrete relation (e.g. structural dependency) between entities/nodes. Further the evolution of systems with hierarchical structure takes place more quickly than with non-hierarchical ones of similar size. Given the composition of complex systems as sub-systems, the interaction within sub-systems is considered more frequent (and often more stable) than among sub-systems. Interactions among sub-systems are fewer, characterized by strongly varying dependency over time as well as typically higher complexity.[2] Techniques from the area of Object-oriented Analysis and Design (OOAD) widely accepted within the software engineering community are introduced by Booch [Boo94] using the terms *Decomposition*, *Abstraction* and *Hierarchy*. In the context of AOSE *Hierarchy* is substituted by *Organisation* to reduce the emphasis on hierarchical dependencies (as suggested by Wooldridge and Jennings [JW00] – and certainly supported by Bunge's understanding). The core characteristics are recalled at this point:

- *Decomposition* – Large problems are divided into sub-problems of manageable size with a highest possible degree of isolation from other (sub-)problems to limit the designer's scope.
- *Abstraction* – Hiding unnecessary details from the designer is beneficial to limit scope at a given time in order to focus on core problems. Constraining the level of detail not only removes disturbing or confusing elements but also enforces a decision on relevance of every element, resulting in a improved understanding of the problem.
- *Organisation* – Identification and maintaining the relationships between different components helps to decide whether they should be perceived as a single unit (i.e. hiding the internal dynamics in the spirit of abstraction) or clearly identifying their relationships and interactions but leave them separated – depending on level of granularity as well as reusability by other

[2]Here it should be mentioned that Simon's understanding is fairly consistent with Bunge (apart from his use of the term hierarchy). However, the latter provides a more in-depth insight into system properties such as coherence.

components not directly related to the sub-system (e.g. loose coupling in the context of open systems).

The agent concept is suitable as a unit of analysis/design to apply effectively the same techniques. However, agent properties can provide added value.

Agents qualify to support the decomposition of problems similar to objects. However, in complex systems the anticipation of all possible solutions is impossible. As sub-problems are decomposed into agents which are themselves not fully able to provide all necessary functionality to ultimately solve their problem, firstly a certain autonomy in order to find a provider for this functionality is necessary. Secondly, the expressiveness involved in interactions – provided by knowledge-level agent communication languages – levers the issue of modelling interaction (and debugging/maintaining it) on a semantic rather than syntactic level. As the flexibility at runtime is promoted, the coordination of the system is necessarily handed over to its agents, rather than maintaining centralized control structures. As a consequence complex agent-based systems – perhaps a characteristic most similar to human societies – lack a deterministic system state. Even if agents provide state information on a regular basis, this information will only represent their individual state at a discrete point in time and most likely only represents an application-related subset of the complete state information[3].

Abstractions in concepts hide a concept's constituents and suppress it from the modeller's view in order to reduce the perceived complexity. Interaction between the abstracting concept and its constituents in the context of agent-based systems can be of considerably high level. The interactions in fact often show characteristic patterns such as coordination, cooperation or negotiation in order to achieve this value-adding functionality. Agent-related technology provides the means for the representation, such as agent communication languages which – superficially introduced earlier (in subsection 2.2.3) – act on the knowledge level. Given a standardization on the syntactic level along with generalized semantic interaction patterns, this reduces the need to deal with lower-level technical issues. Instead

[3]This argument is driven from the conceptual side: The state representation is *given* by the agent, rather than *read* by some coordinating entity. Agent-based systems – if designed as such – could very well undermine the agent concept and directly access its state – which depending on application field is useful or even necessary (e.g. reporting in simulations).

developers can concentrate, in a more productive manner, on problem-related semantic issues. Although the expressiveness of ACLs – as well as internal representation of higher-level agents – inherently increases complexity and enforces steep learning curves, well-specified or already provided interaction patterns allow a quick and reliable representation of auctions or other forms of negotiations. Those patterns can be expressed using the aforementioned interaction protocols to model courses of interaction (conversations).

Probably the biggest benefit in using agent-based concepts is gained when considering their ability to act in organizations and support organisational change (with regards to members as well as structure) at runtime as organisations are "first-class entities in agent-based systems" [JW00]. This again is related to the self-organisation nature of agent-based systems and the decentralization implied (recalling Bunge's understanding). A potential deriving from this is the demand-related growth of functionality at runtime (as suggested in subsection 2.1.3).

3.1.2 Comparing Agents and Objects

Although agents and objects use a similar approach to systems modelling providing powerful means of decomposition, abstraction and organisation a more in-depth view on their differences cannot be neglected in order to advocate the uptake of agents. Numerous discussions are documented (for example Jennings and Wooldridge [JW00], Wooldridge [Woo09], van Dyke Parunak [Par99] and Odell [Ode02]). The development of programming paradigms shows an increasing drive towards abstraction from the machine level; principles like encapsulation and modularisation of code are increasingly used and successively extended. This does not only concern class-level properties and methods but even goes further towards diversity on instance-level and the encapsulation of complex behaviour which can eventually be expressed as rules, or – applying a mental understanding – by concepts such as beliefs and goals. Interesting core differences between agents and objects are briefly described as following, mostly relying on Odell's comparison [Ode02] with focus on features related to agent autonomy and interaction:

- Agents, in contrast to objects, typically involve in (but are not confined to) *asynchronous communication.*

- Messages between objects can only carry call or response for a single method (and parameters) as opposed to agents which allow more flexible message and/or language models of *higher expressiveness*.
- Agents can involve in (potentially multiple and parallel) *conversations*.
- Agent *organisations are flexible*. This can range from a centralized and as such comparatively controlled organization - as with objects - towards more decentralized organisational structures[4]. This feature is central to its popularity for group representations in MAS applications such as simulations.
- Agents allow *dynamic and/or multiple classification* and are not confined to their according 'classes'[5] for a life-time (such as objects). This allows agents to 'play' multiple *roles* which themselves can change over the agents' lifetime; new roles can be added or existing can be disposed. As such the agent concept is significantly more expressive than 'out of the box' objects to describe relations in arbitrary kinds of systems.
- Agents may *develop instance-level features* at runtime. This reflects the potential of agents to learn and potentially change their behaviour. In contrast, objects of the same class will not change their feature set (i.e. properties or methods) at runtime.
- Agents are *small in impact, time and scope*. The impact of a failing agent within a MAS has typically limited impact as interactions are characterized by loose coupling and the need for fault-tolerating mechanisms. In OO systems usually an exception would be raised if an object (or a reference to it) is lost which is likely to stop the operation of the whole system. Impacts are comparatively unproblematic as full agent life-cycles (including death) need to be considered at runtime (small in time). Dereferencing an object and releasing its memory (or waiting for it to be garbage collected) at runtime will eventually have the same effect. But it needs to be carefully coordinated and designed in conjunction with the overall system behaviour and is not as 'natural' as for agents.

[4]At this place the extreme case of strongly autonomous agents not adhering to any organisation (in the structural sense) is not elaborated.

[5]The terminology used is taken from OO.

The limited scope of agents is fairly comparable with the object principle in which objects hold local knowledge and immediate references to other objects but have no representation for the whole system state.

- Agents can show *emergent behaviour*. This is driven by the idea to have an own thread of control respectively autonomy in order to engage with other agents and/or the environment. The 'out of the box' object concept lacks the thread of control, thus the ability to show problem-related meaningful emergence is limited.[6]

Given this comparison and the motivation of AOSE to model systems in general the perception of agent-orientation as succeeding paradigm – in contrast to object-orientation – might arise. Suggestions of this kind should treated with care; the intention is rather to indicate fields in which agents seem more suitable than object-oriented modelling. As such factors like system size, determinism, degree of concurrency, openness and autonomy of its components need to be considered before blindly adopting one or the other approach.

The following Table 3.1 lists several categories of criteria indicating the suitability of agent-oriented modelling approaches for information systems from a rather practical standpoint. Categories predominantly or intuitively applicable for either one development principle (such as reasoning capabilities) are not reflected in this comparison. The characteristics are structured by categories of aspects all of which are briefly introduced in the following.

Behavioural Aspects capture both executional and motivational autonomy. Both Object-orientation as well as Agent-orientation support a basic executional autonomy, in the case of objects (depending on implementation language) in the notion of threads. Goal directedness in contrast, subsumed under motivational autonomy is not available to objects. Equally, pure object systems do not show (meaningful) emergence. Agent-based systems are (if not explicitly modelled in a synchronous manner) non-deterministic (which makes them appealing to generate emergence[7]).

[6] In their nature as multiple objects (and their likeliness for interaction/referencing) they will ultimately show some sort of emergence (e.g. mean memory use of object, ...). Questionable, however, is the usefulness.

[7] In fact determinism and emergence show an inverse relation. As they are not opposite concepts their separate listing is still useful.

Criterion	Object-orientation (OO)	Agent-orientation (AO)
Behavioural Aspects		
Executional Autonomy	O	+
Motivational Autonomy	–	+
Emergence	–	+
Determinism	+	–
Structural Aspects		
Large System Size	O	+
Small Memory Footprint	+	O
Concurrency	–	+
Interoperation Aspects		
Distributedness	O	+
Syntactic Openness	+	+
Semantic Openness	–	+
Maintenance Aspects		
Hot Swap of Elements	–	+
Debugging	+	–

– represents relatively weak support by according paradigm
O represents a medium level of support
+ represents relatively strong support
Example: AO is not specifically focused on development of applications prioritizing small memory footprint.

TABLE 3.1: Criteria for selection of Agent-oriented approaches vs. Object-oriented approaches for System development

If this property should be controllable (such as in production environments) or do not allow multi-threading, an object-oriented approach can be the 'safer' choice.

For *Structural Aspects* the paradigms show a similar trade-off. Agent-based systems generally have decentralized control which suggests their use for large-scale systems with the consideration of scalability. As the overhead for taking up agent-based technology (learning curve, additional tools but also memory overhead) can be significant, this may limit its suitability for small systems. Concurrent execution, in contrast, is one of the builtin strengths of agents.

Aspects of interoperation include distributedness which is feature of most agent-based system, and levels of 'openness'. If syntactic openness, e.g. on network transport level is of relevance, both technologies are equally useful. When communicating on knowledge-level, agent-based systems show clear advantages against classical OO solutions.

The aspect of hot swapping refers to *Maintenance Aspects*. The exchange of entities at runtime is unproblematic for agent-based systems as of their explicit consideration of lifecycles. Objects typically have greater impact on the system as a whole (see the listing of differences above), their exchange at runtime is not part of their conceptual nature. Last aspect mentioned is debugging which is of practical nature but inherent part of software development. Object-oriented systems and their deterministic nature ease debugging, while the decentralized nature of agent-based systems can make this a rather challenging task.

The provided comparison supports the decision for a potential use of agent-orientation for a particular problem. However, none of the listed criteria should be seen as KO criterion for either paradigm but rather a matter of priorities set by the application developer for the resulting system.[8]

3.1.3 Criticism

AOSE is surrounded by a considerably active community for more than a decade now. However, the achievement of the major goal – broad adoption of MAS in everyday software engineering – remains arguable. Although several successful implementations are documented, those are often driven by developers dedicated to the agent-oriented development principles with focus on rather specialized industrial solutions[9]; considering agent-oriented modelling in the standard toolbox of software engineering is yet farfetched. Realizing this, some problems related to this should be outlined.[10]

Yet agents or MAS suffer from their origin in the field of AI and the intimately linked problem of their non-deterministic nature, issues which are not only paradigm-related but partially self-induced.

First, the field itself is divided about its understanding of agents, to a degree in which it eventually becomes common sense to be asked what 'your' agent understanding actually is. It can be argued that researchers favouring the strong notion

[8] In fact both approaches can be generally used to model all mentioned aspects, especially when considering agents as a specialization of objects (see subsection 2.2.2) – however, with different advantages.

[9] Examples for those companies are Whitestein Technologies and Agentis Software.

[10] This part represents a blended approach of literature-backed argumentation along with personal impression.

often confine themselves to a comparatively strict agent understanding and in fact sacrifice significant potential of agents (multi-level approaches, limited heterogeneity, focus on direct communication) – understood here as a fairly 'narrow' notion. Others adhering to a 'wide' notion leave themselves a by far more extensive modeling realm, albeit trading the homogeneity of agents. Additionally to this, the AOSE community has largely been blurred by the attractive concept of cognitive agents interacting in high-level conversations. Albeit providing different means of reasoning the question remains if those agents are of practical use, especially with respect to performance[11]. The integration of both low-level agents, systems building on emergence and high-level agents has not been fully exploited as a relevant topic; high-level systems typically remain agent-programming-language-centric (e.g. 3APL [3APa], Jason [BWH07]) while lower-level approaches hardly integrate an interoperation concept for cognitive agents. In fact only micro-agents strongly argue this weak wide position (i.e. seeking the integration with stronger agent notions) in the context of AOSE – and are the key concern in this work.

As a consequence of the split field, Georgeff eventually argues to not even mention the terms 'MAS' or 'agent' while integrating them with architectures which have a fair degree of similarity but are eventually lacking functionality which MAS can contribute [Geo09]. Here to mention would be the field of service-orientation which offers several service levels (e.g. data level services vs. coordinating services). Agents would just fit in the coordinating role, given the natural suitability to handle errors, common for loosely coupled systems, and to find alternatives in a flexible goal-driven manner.

Another aspect of practical relevance is testing and debugging. The lack of broadly accepted methodologies or mechanisms for debugging and testing, certainly complicated by the non-determinism of agent systems, hampers a broad uptake. Eventually developments in this area can take considerable time and are even complicated as the standardization efforts (which could and should have considered those aspects) are slowing down and seem to develop away from the need to implement

[11] At this point it should be referred to an experiment documented in [FNP10], resulting runtimes for simple agent interactions varying by orders.

existing standards[12] – resulting in diverse and hardly unifiable approaches. A focus which is endangered to be lost along with the starving progress of FIPA specifications is the concept of openness.

Another potential not yet fully exploited is the use of goals for manifold purposes, be it modeling (goal decomposition), testing or debugging [Win09]. Eventually, generic goal-oriented testing methodologies mark an important trend in AOSE (see [NPT10]) and are one key to better adoption of MAS in practice [Win09].

3.2 Agent-based Social Simulation

The application fields for agent-based systems vary widely. Many of them however, rather implicitly qualify for the use of agent-based technology, such as robust (e.g. self-repairing systems) and highly concurrent distributed systems (e.g. logistics); agents are one option to implement those. Especially in production systems agent-oriented software comes along blended with other complementary technology such as the better-received and well-standardized service-oriented computing[13], e.g. in shape of agent-based web services.

One explicit stronghold for agent-based systems today is social simulation which considers itself a "killer application" [EEHT07] for MAS.

3.2.1 Heritage of Social Simulation

While the general area of software engineering recognizes MAS as 'yet another paradigm' to model complex systems, the social sciences and economics have come a long way borrowing approaches from disciplines such as physics, known as sociophysics respectively econophysics, to model their problems (see Troitzsch [Tro09b], p.56). Selected approaches[14] include *System Dynamics* which essentially use differential and difference equations to build complete (and closed) models. Historic examples for models of this kind include several 'world models' (Forrester [For71], Meadows et al. [MMR92]). However, modelling with system dynamics – as the

[12] While in the beginning of the 2000's the implementation of FIPA appeared as a must (e.g. JADE, JACK [Win05], 3APL, OPAL – with minor exceptions such as MadKit), the consideration of strong standards has moved to the background (e.g. in Cougaar, Jason).

[13] A good comparison between both paradigms and the related standards is provided by Chao and Griffiths [GC10].

[14] The following overview on historic simulation approaches is based on Gilbert and Troitzsch [GT05] as well as Troitzsch [Tro09b].

case for all equation-based approaches [Fer99] – is only concerned with the macro-level, thus does not allow the consideration of complex behaviour of individuals and hardly integrates qualitative aspects. The lack of modelling on the micro-level along with the lack of knowledge about actual dependencies on the macro-level itself makes such global approaches prone to produce unrealistic results and opens a range for speculation, known as 'trap of tractability' (Doran and Gilbert [DG94]). This is especially of concern when considering non-linear dependencies of model internals (see Gilbert and Troitzsch [GT05] for an extended discussion).

In contrast to this, *Microsimulation* models (or *micro-analytical models*) put individual-based modeling into focus. Although this high-level description would also capture agent-based modeling, the actual approach differs and still lacks the interaction between individuals: In microsimulations a database of individuals (e.g. based on existing data) is created and predefined stochastic rules representing certain processes (such as aging, marrying) are applied to the participating individuals respectively the data sets. The aggregate of these individuals constitutes the macro-level. Limitations of this approach is the typically lacking interaction between individuals and the only unidirectional influence of micro- to macro-level. Especially in the context of sociology (an area from which agent-orientation heavily borrows (see description of levels in subsection 2.2.2)), the ability to model individuals bears significant advantages – not to the extent to model individual behaviour but a better traceability and realistic grounding.

Subsequent approaches, which extend the principle of microsimulation, are concerned with multi-level modelling (such as individuals, households and populations) and allow simple interactions between individuals and introduce mutual influence of different levels. A simulation environment of this kind is MIMOSE (Möhring [MÖ90]) which makes use of the functional programming paradigm and targets towards a more general use than other systems of its time.

The historical trend of social simulation[15] shows the increasing demand to model both individual actors (micro-level) and observe macro-effects (first-order emergence) along with the potential perception and manipulation of those macro-effects

[15]For more details refer to Gilbert and Troitzsch [GT05].

by cognitive entities on the micro-level (second-order emergence)[16] (see Gilbert and Troitzsch [GT05]; Squazzoni [Squ08]). The use of multi-agent systems for this purpose is thus only but consequent in order not to give up realism when working with formalised models in the context of sociology (see Troitzsch [Tro97]). The *assumed* autonomy of individual agents (in the understanding of the intentional stance), their sociability, their potential situatedness and the non-deterministic nature of MAS (a feature ironically seeked to be controlled by AOSE for the use in production systems) is intentionally exploited and has given MAS a paradigmatic status for modeling and simulation in the social sciences – as Agent-Based Social Simulation (ABSS), and in a wider simulation scope as Multi-Agent-Based Simulation (MABS).

3.2.2 Methodological Aspects

In contrast to other fields, including software engineering, in the area of simulation, especially in the context of the social sciences, general methodologies for the construction of simulation models are hardly found; modellers in this area still exercise a significant amount of freedom but borrow guidelines (such as the validation problems) from other research methods such as experiments. An extreme capturing the essence of the experimental nature of social simulation can be gained when retracing the idea of artificial societies, which Epstein and Axtell see as *laboratories* to "... 'grow' certain social structures in the computer ..."[EA96]. The goal is to create abstract models, in this extreme case purely theoretical models of social structures and dynamics to refine those iteratively until they exhibit characteristics of interest (be it driven by the intention to 'rebuild' societies (and as such not purely artificial) or the observation of unexpected behaviour).

A first and general approach adopted from the field of computer science is Zeigler's Framework for Modeling and Simulation [Zei76], defining the fundamental relationships in simulation systems, the relationship between source system (or real system), experimental frame (the observed objectives of a source system; experimental perspective), the actual derived model and the simulator. Michel et al. outline its suitability for the general case of agent-based simulation [MFD09].

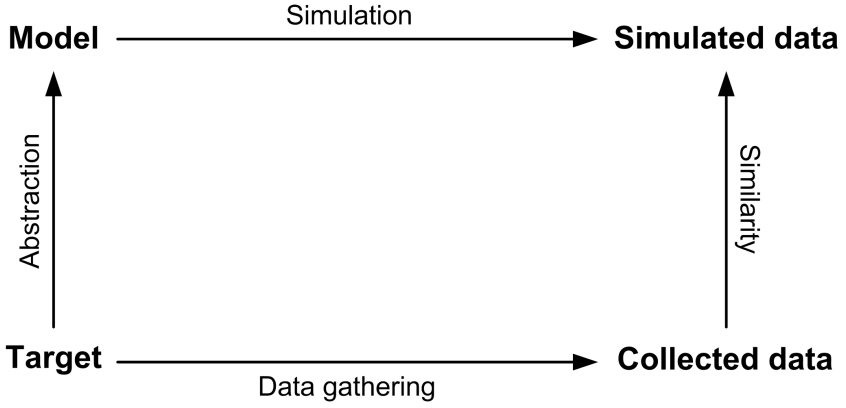
[16] Gilbert [Gil95] suggests that this property discriminates human societies from animal societies.

Another more generally applicable methodology for modelling experiments in AI systems is MAD (Modelling, Analysis and Design), suggested by Cohen [Coh91] as a result of surveying contemporary AI systems. Its level of detail is lower (as compared to the Zeigler framework) and keeps the focus on the implementation perspective.

Fishwick [Fis96] identifies three (triangular) interlinked stages of simulation (in general): Model design, Model execution and finally Model analysis. Downside of this approach is the obviously fluid translation of design model to execution model; the methodology does not sufficiently respect this concern.

A methodological framework born in the context of social simulation which puts a stronger emphasis on the modelling target than the implementation system [ACBR06] is described by Gilbert and Troitzsch [GT05] and includes the following steps:

In first instance, a model of a *target* (the phenomenon of interest) is built. The model is sufficiently simplified, still representing the structure and behaviour of the target but avoiding an unnecessary level of detail. The model – which is inherently dynamic in the context of the social sciences – is executed and produces simulated data which should be tested against gathered data showing the 'real' dynamics of the target.[17] Figure 3.1 outlines the framework described.



Source: Gilbert and Troitzsch [GT05], p.17.

FIGURE 3.1: The logic of simulation as a scientific method

The framework further consists of several steps when building simulation models:

[17] This is certainly only possible if the target is in fact a real system allowing the gathering of such data.

- (1) Definition of the modeling target
- (2) Observations on the target
- (3) Definition of underlying assumptions
- (4) Verification of the model implementation
- (5) Validation of the implemented behaviour
- (6) Sensitivity analysis

The definition of the target (1) describes what behaviour and structure of a real world entity (if the concept is of 'real' nature) is observed (e.g. the intensity of interaction within an artificial society). This is sharpened with the identification of observed/dependent variables (2) which allow measurements of the model behaviour (e.g. kinds of interactions, duration of average interaction, number of interactions, ...). Further, underlying assumptions (3) need to be clear (e.g. observation of a societies without access to means of distance communication).
Assuming a significantly clear model specification, the verification (4) is concerned with the implementation of the specification in a simulation and includes debugging activities. This is particularly problematic as a bug-free implementation can hardly be garanteed (especially when considering the heterogeneity of implementations/simulation environments (see following subsection)). The use of (pseudo-)random number generators demands for special attention when attempting verification (see Gilbert [Gil96]). In the validation step (5) the sufficiently adequate representation of target behaviour is evaluated; the model specification is reviewed. The last step, the sensitivity analysis (6), checks the sensitivity of the simulated data towards changes of the parameter set to establish a good understanding of the model and identify parameters of core relevance.
The actual modelling activity, especially in the context of social sciences, has the character of a *wicked problem*[18] as its quality is hardly verifiable and largely depends on the experience and abilities of the modeller. The later steps support the often necessary iterative refinement of the model. A guideline for a modelling strategy lies in the purpose of the model (see Axelrod [Axe97]): Descriptive models

[18]Some characteristics of wicked problems include the lack of a stopping rule, the problem is often not understood before elaborating a solution, the uniqueness of each problem instance and the lack of a distinct right or wrong solution (see Rittel and Webber [RW73]; Conklin [Con05]).

which support the understanding of a phenomenon of interest focus on *simplicity* while predictive models put an emphasis on *accuracy* – implying a higher level of detail.

For an in-depth description and discussion of the caveats for given modelling steps the reader should refer to Gilbert and Troitzsch [GT05].

3.2.3 On the Gap between MAS for AOSE and ABSS

Having stated that social simulation focuses on core agent properties like autonomy (and as such is largely (and often purely) based on the intentional stance), its perspective deviates from the one taken by most agent researchers. In fact agents in the context of social simulation rather underly a simplistic agent model often assuming some sort of autonomy (in contrast to the more complex understanding of this concept in the agent community (see subsection 2.1.1)). Additionally they often focus only on indirect communication (e.g. Schelling model [Sch71]) – again a feature argued not to be sufficient to constitute social ability of agents (see discussion in subsection 2.1.1).

Given this the application field for social simulation represents a meeting of the spirits from different streams: AI and DIS originate from the idea to shape potentially intelligent software entities, but have increasingly loosened this perspective[19], resulting in higher flexibility. ABSS in contrast is prone to see virtually any object as an agent but increasingly adopts stronger agent notions – in the context of the need for more adequate representation of deliberative capabilities[20], available computing capabilities (capacity for 'more agents'[21]) and modelling mechanisms (exploitation of concurrency in multi-core processors with truely asynchronous agents). Table 3.2 compares characteristics of the different streams.

The differing perspective – from a technical vs. a rather pragmatic view – on agents supports this observation: Researchers from the area of (general purpose)

[19]The increasing consideration of weaker agent notions (such as in MadKit, Jadex and Cougaar) back this observation (see subsection 2.1.3). Additional to this the coupling between (optional) reasoning engine (as in Cougaar) and platform becomes less intimate.

[20]An example includes the demand for stronger cognitive abilities on the micro-level in order to model second-order emergence as well as more 'knowledge-based' direct communication.

[21]The comparison for approaches of social simulation provided by Gilbert and Troitzsch (see [GT05], p.13) indicates a typically limited number of agents ("few") in contrast to other approaches considering a larger amount of entities (such as microsimulation).

Field	**Objective**	**Key driver**	**Agent notion**	**Modelling**
Artificial Intelligence	Modeling cognitive processes in a connectionist approach; agents as part of a collective cognitive unit (in the psychological sense)	Reasoning	strong narrow	agent-centric
Distributed Information Systems / Software Engineering	Helper paradigm to model distributed systems with autonomous and heterogeneous elements	Interaction and (de)composition	strong wide	information system-centric
Social simulation	Understanding the structures and dynamics of individuals, groups and societies in a given application context (e.g. economics, psychology, artificial life) by experimentation, generally accepting multiple levels of causation and effect	Emergence	weak wide	target phenomenon-centric

TABLE 3.2: Comparison table for field-dependent perspectives on MAS

MAS see the *execution* of an agent-centric model of core relevance. They focus on the runtime environment (i.e. simulator). Michel et al. (which belong those the former field) argue that "... few [simulation] MAS works do consider the simulator as a first order entity of the experiment." [MFD09] and criticize the lacking consideration of the MAS paradigm itself by applying researchers (or 'thematicians' as Drogoul et al. [DVM02] name those) at the modeling stage.

The thematicians in turnaround perceive the MAS paradigm as a vehicle to "... engineer models from concepts to something that can be executed on the computer" [Fis95], thus see the model implementation as a mere engineering task. This difference (system-centric vs. target-centric modelling) is already apparent when recalling the short listing of modelling methodologies above[22].

As far as this is concerned, influx from both groups is necessary in modelling activities for each given case: firstly, to get a specific understanding of the agent notion applied, secondly, to keep the gap between target-centric conceptual model and agent-centric design model narrow – in the best case so narrow that the simulation programme itself is a full model of the theory of interest (see Troitzsch [Tro04]). This certainly describes the extreme case where sufficient knowledge about both fields is available or cases where modelling the target phenomenon itself maps well to the paradigm. Still, the wicked problem of modelling is not eased by the fuzzy understanding and use of the term 'model'. Wartofsky [War79] identifies several confusing understandings of models. As such a model can be the actual conceptual model of a target or the largely implicit mental model the modeller develops. Over time modellers develop an increasingly blended understanding and usage of those. Last model type is the data model (i.e. the output of the (implemented and executed) conceptual model) which represents an abstraction from the originating conceptual model by introducing further prerequisites and assumptions. The latter model might in fact not sufficiently capture and represent the content of the original conceptual model.[23]

To complement the view on the tensions between Modelling and Simulation (M&S) and the technical perspective on MAS, another approach on structuring implementations is considered. Michel et al. [MFD09] view developments in the intersection of MAS with M&S as belonging to either one of two groups:

- M&S for MAS
- MAS for M&S

[22]Cohen's approach is rather system-centric, Gilbert and Troitzsch focus on the 'modelling target'.

[23]Further elaboration on those different models is provided by Wartofsky [War79] as well as Edmonds et al. [EEHT07].

The first one includes approaches which deal with modelling of internals of MAS systems, most prominently complex issues such as modelling open interaction with the help constructs of interaction protocols, such as the Contract Net Protocol (Smith [Smi80])[24]. Concepts of this area target towards the improvement of MAS in general; as such works relating to this group tend to be rather self-reflective (with regards to MAS).

On the other hand, MAS for the purpose of Modelling and Simulation, reflect a wide field of different approaches, respectively simulators. Michel et al. identify three categories of simulators, namely the overwhelmingly dominant group of *'one shot' simulators* which are individually developed for a specific simulation case, *domain-dependent simulators* (e.g. Conflict Research: GROWLab [WG06]; Ecology: Echo [HJF97]) and finally *generic simulators*, applicable in various constellations. Examples for the latter ones include Swarm [MRBCL96], Mason [LCRP+05], RePast Simphony [NCV06], NetLogo [Wil99] and SeSam [KP98].[25]

Applying this understanding it becomes clear that nearly all simulation applications belong to the field of MAS for M&S; this differentiation is hardly helpful to describe applications of MABS as of its MAS-centricism. Nevertheless, the classification of simulators helps to structure simulation applications by their degree of specialization. A survey on MAS simulation toolkits which takes a different perspective and structures simulation toolkits by their (increasingly complex) execution models and distributedness is undertaken by Theodoropoulos et al. [TMEL09]. Although not further discussed at this point it supports the trend of increasingly heavier agent notions in agent-based simulations.

To allow a more beneficial differentiation of distinct features multi-agent simulation platforms[26] have in contrast to multi-agent platforms the review of literature from the simulation community seems useful. A comprehensive requirements analysis framework for MABS platforms, suggested by Marietto et al. [MDSC02], is

[24] Michel et al. [MFD09] provide further examples of this kind.

[25] At this point it can be anticipated that the development undertaken in this thesis yields towards this group.

[26] The terms toolkit and platform are interchangeable for the understanding of simulation platforms taken here.

particularly helpful in this context. Their framework, especially the subset of functional requirements [MDSC02], identifies four core facilities:

- *Technological Facilities* describe the supported scheduling techniques, such as equidistant time-stepping and discrete or continuous event scheduling.
- *Domain Facilities* discuss the mechanisms by which agents can be launched (e.g. object, threads) as well as the handling of intentional failures and integrating (controlled and non-controlled) environments.
- *Development Facilities* include means to develop agents (i.e. the supported agent architecture), message handling, organisational structures (including groups and societies) and the support of ontologies.
- *Analysis Facilities* summarize all mechanisms for the observation of (behavioural and cognitive) events and data analysis.

Applying those facilities to general purpose MAS especially two areas are hardly explicitly concerned: Technological facilities such as scheduling, necessary to assure a strong degree of fairness and replicability of results to accept MABS as a credible research tool, are of secondary role in the case of MAS used for AOSE; here operating system level threading fairness seems to be sufficient, given that individual agents eventually get processor time and the required system functionality is delivered[27]. Taking into account reviews of MABS platforms, respectively their rating schemes, helps to indicate particular differences between AOSE and MABS platforms. Railsback et al. [RLJ06] provide an overview on popular platforms but show a less elaborated criteria set in comparison to Tobias and Hofmann [TH04] who provide a very detailed systematic rating system but evaluate (in the meantime) less popular platforms. Groups of criteria include 'Documentation and Support', 'Modelling and Experimentation' as well as 'Modelling options'. Criteria from the category 'Modelling and Experimentation' which are of interest in this context include 'Support for modelling' indicating the degree to which modelling can be automated (ranging from the need to fully implement agents in Java towards the potential to do advanced content-theoretical modelling without

[27]In fact most agent platforms do not consider an explicit fair scheduling at all, merely provide building blocks for the implementer or use it to share operating system threads amongst agents (such as in MadKit).

programming knowledge). Additionally 'Support for simulation control' describes scheduler capabilities (from simple 'running' to advanced features such as manipulation of parameters during runtime or the integration of differential equations for simulation control).

Interesting criteria from the 'Modelling Options' include the number of supported agents as well as their complexity (which is not further explained by Tobias and Hofmann [TH04]), their communication (ranging between no support to efficient processing of complex data exchange processes) as well as the ability to nest agents in the understanding of levels (ranging from the absense of nesting to an unlimited number of levels).

When reviewing the rating for scheduling options it becomes apparent that the application-level[28] scheduling component in fact represents the bottom-line of the feature set of those platforms. Even examples for less comprehensive frameworks, such as Mason which merely provides a simplistic agent understanding with hardly any communication means for agents, the powerful control and visualization capabilities are apparent. The strong emphasis on the control (of experiments) is hardly of relevance in MAS for AOSE.

A concern not captured by the requirements framework or evaluation schemes, but certainly of concern in contrast to general purpose MAS, is the consideration of (pseudo-)random number generators – to firstly have a (nearly) equal distribution of numbers, secondly, to be able to replicate number sequences in different rounds to ease testing/debugging or, again, reproduce results.

The other area of concern are analysis facilities. Observation of distinct events can often be monitored in general purpose MAS but rather fall in the category of debugging mechanisms, often in the shape of agents[29] or hooks for external monitoring (such as in MadKit). As of the high performance penalty involved, the use of those facilities at runtime is hardly desirable which sharply contrasts to simulation systems where those are constantly available. Data analysis mechanisms (apart from simple logging), especially considering ex-post analysis and reporting,

[28]Scheduling on application level is particularly relevant to capture the application/platform semantics.

[29]Examples include the SnifferAgent in JADE and the MindInspector in Jason.

are not of concern in general purpose MAS.

An aspect drilling into the core of platforms is the strength of the supported agent concepts and communication abilities. Mapping the scales of the comparison framework provided by Tobias and Hofmann [TH04] to MAS for AOSE makes apparent that the 'Modelling options' for AOSE-related multi-agent platforms range on the strong end. AOSE-related agent platforms typically allow a strong complexity of agent internals and provide support for meaningful direct interaction. A particular advantage for organisational MAS in contrast to coarse-grained agent platforms for AOSE – as far as introduced at an earlier stage – is their builtin support for nested agents (as of their multiple levels). This, along with powerful agent modelling capabilities and strong communication means, makes the use of the meta-model implementations appealing for simulation purposes as of the strong relation to society modelling demands in simulation packages. Retracing the results for two still actively used platforms in Tobias and Hofmann's comparison (i.e. RePast and Swarm) it can be seen that the strength of interaction mechanisms significantly falls back in contrast to general purpose MAS[30] despite their else strong means of agent modelling. This observation, along with the generally weaker notion of agents, indicates that agents in simulations put a stronger emphasis on indirect communication. Malsch [MSK+07] suggests that direct communication is hardly of concern in social simulations. Instead communication even goes to the extent of communicating in a telepathic manner (see Hutchins and Hazelhurst [HH95]) – directly reading (accessing) the other agent's mind. A consequent parallel trend is the use of general purpose platforms from the AOSE field for simulation, particular when modelling is limited by lacking support for particular features of interest, such as reasoning mechanisms or, as indicated before, knowledge-level communication. Apart from this the increasing available computing power now allows the use of larger numbers of conceptually more powerful agents. Jason is a candidate to provide features 'compatible' with

[30] In fact they only allows rudimentary interaction patterns with additional implementation demand (see Tobias and Hofmann [TH04]).

social simulation, such as a synchronous scheduling[31] along with its powerful reasoning mechanism and the provision of KQML as communication language (see Bordini and Hübner [BH09]).[32] An example for a simulation using BDI-level features is given by Hofstede et al. [HJV09]. This general trend will presumably continue when considering the increasing interest for the issue of (emergence of) communication in the social simulation community (see Troitzsch [Tro09b]).

Not explicitly emphasized by Marietto et al. but a logical consequence of the mostly indirect communication in social simulations is the role of the environment[33]. Its often increased relevance in the context of simulation can be seen for three reasons. Firstly, the often limited degree of agency (see Table 3.2 in its context) enforces a stronger use of indirect communication[34]. Secondly, the environment represents a field of constraints/conditions under which agents exist and act[35]. 'Conventional' MAS platforms often imply (but do not enforce) the embeddedness in their 'natural' environment (information systems or networks such as the internet) as their paradigm is driven by the aspired openness; simulation platforms used as 'laboratories' (as envisioned by the founders of the social simulation discipline) typically need to constrain the openness of their environment (in short: model them as closed systems) in order to ensure replication of a simulation, and as such put more emphasis on a clearly defined environment concept. The last reason for the explicit consideration of environments is the degree of realism in simulated systems (in general), ranging from purely artificial environments (for the development of a controlled initial behaviour) with the potential for transitions to increasingly realistic environments (to test the realism of modelled agent behaviour (e.g. traffic

[31] Braubach et al. [BPL+06] suggest this as the key mechanism to use conventional agent platforms in the context of simulations.

[32] By running Jason in conjunction with JADE, agents implemented in Jason are able to interact with other 'FIPA-speaking' agents.

[33] Tobias and Hofmann mention the modelling potential for the environment (e.g. modelling of spatial aspects) in the category of 'Modelling options'.

[34] This shall be seen independent from technical implications by which communication can always be seen as indirect (when considering the *infrastructural environment* as introduced in subsection 2.2.2).

[35] For a more detailed discussion on the role of the environment refer to Troitzsch [Tro09a].

simulations, agent behaviour in virtual reality))[36]. The latter point, the application of different environments for simulated models, suggests a development into an interesting direction. The continuously increasing processing capabilities on mobile devices – available to model implementations – allows to 'take them along' and realizes an ubiquitous embedding in real environments on an individual level. Summing up, the key differences between general purpose MAS for AOSE and MABS (and particular social simulation) outlined in this context are *scheduling to ensure fair and replicable simulation results*, *explicit analysis and reporting features* as well as a (mostly) *stronger emphasis on a controlled environment*. Agents in simulation packages rather rely on indirect communication mechanisms while AOSE-oriented MAS put a stronger emphasis on direct communication. Beyond that the general agent concept in platforms dedicated to the use with simulations is of weaker nature. Exceptions of the latter make use of AOSE platforms for modelling of simulations. Table 3.3 sums up the core differences between multi-agent platforms for MABS and AOSE.

Platform Aspect	**MABS-centric Platforms**	**AOSE-centric Platforms**
Fair Scheduling	+	–
Random Number Generation	+	–
Analysis and Reporting Mechanisms/Visualization	+	– / O
Direct communication	– / O	+
Indirect communication	+	O
Complexity of agent model	– / O	+
Importance of support for large numbers of agents	+	O
Importance of the Environment	+	O
Open systems	–	+

– represents relatively weak importance for the according platform type
O represents a medium level of importance
+ represents relatively strong importance
multiple symbols indicate a 'fuzziness' with regards to the relevance in platforms

TABLE 3.3: Feature support in Multi-agent Simulation Platforms in contrast to general purpose Multi-agent Platforms

[36]Michel et al. structure the possible combinations of realism in both simulation model and its environment (see Michel et al. [MFD09], p.33ff.).

3.2.4 Problems in Social Simulation

At this point the concept of social simulation has been briefly introduced and differences of (the wider field of) MABS in contrast to classical MAS discussed. Still, despite the increasing application of MAS in social simulation[37] certain issues persist – and partially lie in the nature of the interdisciplinary field.

As stated by Michel et al. [MFD09] the variety of available tools harms reproduction of simulation results and in consequence harms social simulations as a credible discipline as such. Although the difference of the approaches is partially rooted in the different sciences involved, key reason remains the gap between the MAS paradigm and the modelling tools.

However, the issue of replication is a known problem in the social simulation community. Practical measures taken against this is the electronic submission of models along with publications[38]. This establishes a (though tool-dependent) reproducibility of results, allowing third party reviews without being limited by often incomplete specifications in the actual paper (as stated by Heath et al. [HHC09]). This however, does not solve the issue of verification of the toolkit used (which would eventually require a guarantee by its developer in turnaround).

Given that a unified view is not even reached within the – in contrast to the social simulation experimenters – comparatively homogeneous MAS research field (with its members typically deriving from computer science-related disciplines) this expectance should not be raised towards the dispersed disciplines of experimenters using agent-based simulation tools. Again to be reinforced, when arguing for a unification of the understanding of MAS in different fields, this is often done from a MAS-centric standpoint against a model-centric user community – which again applies a different understanding of the MAS paradigm. In consequence, this gap is unlikely to be fully overcome, neither from MAS nor simulation side.

Another issue harming the replicability of simulations – given that documentation is sufficiently precise – is the *engineering divergence phenomenon* [MGF04] which describes the potential or even likeliness that implementations, based on a unique

[37] The survey provided by Heath et al. [HHC09] shows a clearly increasing trend in published simulation results.

[38] An example taking this approach is the electronically published Journal for Artificial Societies and Social Simulation (JASSS) [JASc].

model, divert based on implementation language (e.g. expressiveness) and implementer (e.g. thematician vs. computer scientist).[39]

Implicitly supporting this, Heath et al. criticize that simulation models are often based on numerous assumptions and raise the demand that published models need to be sufficiently validated and fully documented[40]. In this context the lacking methodological consense represents a drawback.

Whatever view taken on MAS it must be emphasized that in the area of simulation in general and social simulation in specific self-criticism certainly should not include lack of adoption[41] (as in the case of AOSE). Rather issues such as reproduction of results or development of modelling patterns (or 'best practices') are of concern. Especially in social simulation the field of users is diverse – making commonly accepted methodologies tackling those purposes only harder to establish.

The suggestion that mentioned drawbacks of social simulation will limit its uptake and existence as an own scientific discipline can be rejected when retracing the increasing interest, particularly but not only in the field of economics. General stimulus for agent-based simulation in the latter field is driven by the recent economic crisis (see 'The Economist' [Eco]) but also the realization that the complexity and fuzziness of dependencies clearly spans across the wider field of (not only) social sciences (as indicated by Ball [Bal10]). This leaves agent-based social simulation as the only applicable option[42] to provide increasingly comprehensive models[43] to capture a wide range of real-life phenomena.

[39]Michel et al. [MFD09] specifically elaborate on the violation of the important hypothesis (in Zeigler's Framework for M&S) that the results of an model must not depend on the specific implementation (i.e. not exactly one implementation but any reimplementation (on another platform or in another language)).

[40]Assuming that the implementation represents the model (and thus successfully passed the verification step as per Gilbert/Troitzsch methodology), this demand can be achieved in most simulation toolkits (using code annotations and comments).

[41]Heath et al. [HHC09] document a significant increment of simulation-related journal publications.

[42]Drawbacks of purely mathematical approaches to 'total models' (e.g. System Dynamics) have been mentioned in subsection 3.2.1.

[43]The yet most comprehensive model of an economy has been realized in the context of the Eurace project [DVD08] which considers numerous markets (e.g. labour, goods, finance) and includes several million agents.

Chapter 4

Concurrency models of relevant Technologies

The previous sections of this work have been fully dedicated to agent-related foundations. This – considerably shorter – section provides an introduction into several technologies potentially interesting for the integration into the micro-agent framework. Key focus is on the concurrency models/handling mechanisms provided with those which make their consideration for multi-agent systems attractive.

4.1 Concurrency

Before introducing the relevant technology, a short overview on concurrency shall be provided. Upfront however is the question why we have to deal with concurrency as such. The identifiable drivers are in fact twofold, one being the hardware architecture market, the second our area of concern, agent-based systems.

4.1.1 On Concurrent Computing and its Relevance

In 2005 Sutter stated "The free lunch is over" [Sut05] and critically analyzed the changing market for CPUs whose trend of every-rising clock speeds he considered the "free performance lunch" [Sut05] for any available application. The development of this trend has eventually reached its peak as of physical constraints

(power consumption, heat dissemination and current leakage) and made manufacturers switch to alternative approaches whose currently prevailing one[1] is the notion of multiple processing units on a die – multi-core CPUs. As a consequence a concurrent layout of applications is imperative to exploit the full performance potential but also to write well-performing applications in general (as performance gains will be realized by further increasing numbers of cores rather than the quite stagnant current clock speeds).

The concept of autonomy in agent-based systems implicitly incorporates the idea of concurrency; considering their loose coupling an explicit 'programming' of concurrency in MAS applications should in fact not be necessary at all. However, in practice – and depending on the kind of MAS application – handling of concurrency still is an implementation-level concern harming a consistent 'thinking in agents' – especially when accessing common resources such as an environment. To avoid the complexity associated with this, platforms either escape to the very safe ground of sequential execution (as most simulation platforms and (optionally) Jason), introduce strong abstractions for communication (including a high performance penalty (e.g. JADE and OPAL)) or delegate it to the user (e.g. legacy OPAL micro-agents).

Given the understanding for the need to consider concurrency in the context of contemporary software engineering, the usage of this concept in prior sections of this work needs to be backed with an unambigious understanding: In this context *Concurrency* is understood as a property which describes the ability to process multiple tasks/programmes in a *seemingly* or *factually* parallel manner. Here the important but unsharp distinction from parallel computing should be drawn. Parallel computing refers to tasks which are actually executed at the same time – be it multi-core, multi-processor or (coordinated) distributed systems.[2] Although this may actually be the case, most often – and most intuitive in the case of single-processor machines – the execution of tasks is interleaved in a sequential manner, giving the perception of parallel processing. In consequence the scheduling of the

[1] Hyperthreading – an earlier trend in processor architectures – extends regular CPUs by additional registers to store virtually parallel executed instructions but does not allow actual parallel computation.

[2] A valuable source on concurrent and distributed computing is Ben-Ari [BA06].

execution is delegated to the system scheduler. Parallel computing has a stronger link to processor architectures while concurrent computing refers to concerns of handling multiple (often interdependent) tasks in programs, thus has stronger software engineering links. Although parallel computing defines numerous subtypes of parallelism[3], in this context the major difference shall be 'seemingly parallel' (= concurrent) and 'factually parallel' processing.

Programmes, typically developed as independent (however, not isolated, i.e. they may communicate with other programmes) units, run as individual processes which hold their own resources. Within a process multiple threads of execution (threads) have access to the (then) shared resources and produce the concurrent behaviour of this programme. Some (predominantly high-level) languages[4] introduce the notion of threads to model this in-programme concurrency; they represent the bottom-line of concurrency discussed in this work.[5] Key difference is thus the necessity to model access to shared memory on thread level which represents the core problem to be tackled in this work when talking about concurrency.

4.1.2 Concurrency Handling Mechanisms

A problem of stronger relevance than the concept of concurrency itself is the trade-off between exploitation of scalability effects by a concurrent layout of applications (seeking for fully concurrent and thus independent software units) versus coordination of the access to state (which is necessary to provide the coordinated overall system functionality). Coordination mechanisms from both opposing fields are described in the following.

In order to achieve *shared memory communication* – the yet typical approach in conventional object-oriented systems – access to shared memory needs to be synchronized using locking mechanisms such as mutexes (marking 'critical sections'), (counting) semaphores (see Dijkstra [Dij65]) or monitors. Facilities of this kind are found in languages such as Java (or can be modelled from existing mechanisms).

[3] See [Wik] for a broad overview.

[4] Examples include Java [AG99], .NET [NET] as well as the next release of the C++ standard [Nex10].

[5] For further discussion of formal aspects and state models the reader is referred to Magee and Kramer [MK06].

They can allow efficient handling of concurrency as the developer has the full control within his code. The downside of this approach is similarly the fact that the appropriate use is left to the developer, and as such essentially depends on the skills of the developer (Example for simple problem: Wrong choice of locking granularity can easily lead to deadlock.). Especially in the context of object-oriented languages this is considered harmful as object-oriented modelling per se does not respect the multiple threads of execution (as stated by Ousterhout [Ous96]). Threads break the rule of abstraction between different modules, interdependencies in combination with non-deterministic scheduling of threads can lead to hardly identifiable deadlocks or unsynchronized access. This harms reusability and portability of code across different platforms[6], the lack of predictability demands for careful inspection during testing.[7]

More recently discussed approaches[8] to handle shared memory in the context of concurrent computing is *Software Transactional Memory* which effectively seeks to extend the concept of transactions known from the area of Database Management Systems (DBMS) to programming languages.[9] Similar to the relief of the application developer from memory management by the delegation of memory management to the programming languages (such as in Java [AG99] or Haskell [Has]), transactional memory relieves the application developer from concurrency management, thus making the handling of concurrency a system-level concern.[10]

Key concerns in STMs are the problems of *versioning*, *conflict detection* and *nested transactions* (see Adl-Tabatabai et al. [ATKS07]). In case of the first, the concern is how to manage rollback of state changes. Decisions in the context of versioning

[6] Although this is not generally the case for Java applications, application behaviour can significantly change depending on the degree of implementation. An example for such issues is Thread.yield() – usable to induce a more cooperative behaviour of threads – as of its varying implementation on operating system scheduler level (see [Thr]).

[7] Here the developer should be similarly sceptic as with the use of pseudo-random number generators in simulation applications (see subsection 3.2.2).

[8] Although the origin roots back to 1986 (see Knight [Kni86]), the concept of purely software-based transactional memory has been elaborated in 1997 (see Shavit and Touitou [ST97]).

[9] Here to mention is the fact, that the durability property of the ACID (Atomicity, Isolation, Consistency and Durability) properties is left to the actual implementation/language as the shared memory is typically maintained in the heap, not on the disk (or at most virtually in shape of swap memory).

[10] Typical approach is to annotate according code sections and read and write to those without further (or minor) restrictions.

include the consideration of the size of possible transactions (i.e. the size of the necessary buffers maintained as additional memory overhead) as well as the degree of optimism of the writing strategy.

Versioning is realized as *eager versioning* where changes are written immediately (thus achieving performance advantage when successfully committing but performance penalty on roll-back) or as *lazy versioning* in which case all changes are kept in a buffer until the final commit. In the latter case committing the changes is slower; the roll-back is fast.

Conflict detection has to deal with the issue of granularity and includes risk of false detections if conflict detection works too coarse-grained (e.g. concurrent manipulation of object fields while level of versioning granularity can only capture object as a whole, thus cannot resolve conflicts inside an object) vs. the performance penalty when considering a fine-grained detection (e.g. on bit-level).

The handling of nested transactions is imperative when considering to completely relieve the application developer from concurrency concerns (which is intuitive when thinking about the composition principles in the context of object-oriented programming – and in consequence agent-oriented programming). Especially when accessing third party libraries the internals of those (and their use of STMs) should not be of concern to the developer; different data structures might themselves be backed by STMs[11].

The consequently opposing approach to communication via shared memory is the idea not to share anything at all ('share nothing') but to rely on *Message Passing* between software components. In fact message passing always plays a significant role in object-orientation when accessing functionality of different objects as object-oriented programming languages such as Java or Smalltalk [Sma98] rely on synchronous message passing. This pure approach remains only suitable for single-threaded applications as the method call to other objects is firstly blocking and secondly assumes the synchronous representation of data in each object

[11] A data structure showing STM-like patterns is the ConcurrentHashMap implementation in Java which garantees consistency at any time and is segmented to allow concurrent write access [IBMb].

at any time. When dealing with concurrency those two issues are hardly suitable nor can be fulfilled. Scalability effects (as of Amdahl's law[12]) can hardly be achieved when considering the blocking call of methods on other objects. Exception is the consideration of inherently simple objects which do not hold state but expose short-running methods for external use. Especially when holding state a synchronicity of state representation in different objects can hardly be assumed if using multiple threads of execution.

The use of asynchronous message passing to allow non-blocking communication between different software entities – and essentially rooted in the area of distributed systems – both eliminates the necessity to consistently represent state and to allow scalability effects as of its non-blocking nature. A pattern typically associated with asynchronous message passing in the context of software engineering is the actor pattern (as introduced by Hewitt et al. [HBS73] and extended by Agha [Agh86]). Its use has recently been popularized by the increasingly recognized functional inspired but multi-paradigmatic Erlang [AVWW96] programming language, originally developed by the Eriksson telecommunication company to develop robust decentralized systems. However, in the meantime concepts and implementations of Asynchronous Message Passing are available for various programming languages traditionally focusing on lock-based concurrency handling mechanisms such as Java. Criteria to qualify implementations as consistent with the actor pattern include [Agh86]:

- *Encapsulation* of both internal state (of actors) from direct external access (*state encapsulation*) as well as ensuring that messages can only be read by one actor at a time (*safe messaging*). The latter typically results in message copying instead of passing an in-memory reference.
- *Fair Scheduling* includes the idea that actors should be scheduled in a fair manner, i.e. provide fair access to processor resources[13].

[12] Amdahl's law effectively states that a potential speedup achieved by parallelizing a programme is limited by the fraction of the remaining sequentially executed code.

[13] This does not imply that actors need to get an equal amount of processor time – this is entirely dependent on the scheduling means (e.g. preemptive scheduling versus cooperative scheduling) – but equal opportunity to execute their logic.

- *Location Transparency* refers to the principle that an actor's name/address should be immutable and independent from its actual location, freeing its user from any location-related concerns. This is especially helpful in conjunction with the final property, mobility.
- *Mobility* describes the ability of actors to change their location of execution at runtime which comes in flavours of weak mobility – only comprising of the actual code which is reinstantiated on the target node – and strong mobility which also considers the transmission of state allowing processing without discontinuation during the movement between nodes[14].

The obvious downside of the fully concurrent application modelling approach is the fact that the determination of an overall system state becomes a non-trivial task as of its inherent decentralization (in contrast to the shared memory coordination mentioned above). The semantic criteria of the actor pattern give a spectrum of properties to consider when applying asynchronous message passing, independent from the degree to which those properties should be fulfilled.
The mentioned concurrency handling mechanisms show the range of possible approaches to tackle coordination in concurrent systems and the consequences – using indirect communication via shared memory and as such effectively centrally coordinated system state versus the use of messages as a medium to coordinate fully decentralized state.
Figure 4.1 provides a schematic overview of the coordination means in concurrent systems discussed in this subsection.

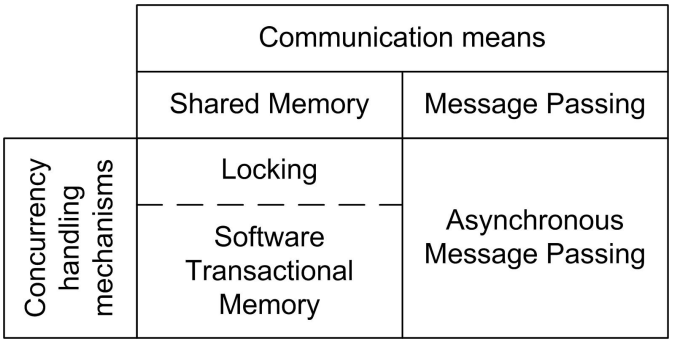


FIGURE 4.1: Concurrency handling mechanisms

[14]This concept has already been discussed in the context of agent properties (see subsection 2.1.1).

4.2 Technologies in the Intersection of AOSE and Concurrent Computing

This section presents contemporary technologies which are appealing from both the perspective of agent-based systems – and in specific the direction taken in this work – as well as of their representative roles with regards to the mentioned concurrency handling approaches.

4.2.1 Clojure

In the stream of porting various languages to the Java Virtual Machine (JVM) – as a de facto standard for platform-independent code execution – a recent move is the introduction of languages of different programming paradigms. Those exploit the potential of the virtual machine while avoiding to be confined to Java as a programming language (and along with this the object-oriented programming paradigm). An example for this trend is the (factually multiparadigmatic) language Clojure [Hic10a], a LISP dialect allowing the developer to interactively develop applications with the LISP-typical Read-Eval(uate)-Print-Loop (REPL) which is fully interoperable with the Java programming language (and as such allows runtime access to Java functionality). Along with this come the principles of the functional programming paradigm. Those include the use of higher-order functions (i.e. functions which take functions as input and/or output), the avoidance of side-effects by emphasizing recursion instead of iteration and the avoidance of mutable state as far as useful. The stronger emphasis on functions in favour of state and promotion of immutable state makes functional languages specifically suitable for concurrent processing. In this area Clojure can provide significant handling improvements over Java's conventional lock-based strategies. To limit the effects of the memory overhead used for the predominant use of immutable data structures, lazy initialization delays the evaluation of functions to the time of the explicit request for a result.

As of its nature as LISP dialect the Clojure syntax is based on symbolic expressions (S-Expressions) allowing strong parentheses-based nesting capabilities which can lead to a significantly reduced amount of code compared to languages such as

Java. The development of prototypes – but not necessarily the maintenance – is eased by its dynamic type system.

Apart from the lack of functional features in Java itself, the further motivation for the consideration of Clojure is its more powerful (and more automated) means of concurrency handling. In order to achieve this the manipulation of mutable state can be backed by the inbuilt STM. Clojure hides those mechanisms nearly completely (apart from explicitly annotating the code sequences which should be processed as a transaction) from the developer but offers different mechanisms of varying strength to achieve this:

- *Vars* encapsulate state which is changed in a thread-local manner, i.e. changes are only valid for the manipulating thread. As the manipulation only affects isolated threads, a backing by the STM is needless.
- *Transactional References (Refs)* allow the "coordinated, synchronous change of multiple locations" [Hic10b]. Changes to all references in the transaction appear to happen at the same time for all potentially accessing threads.
- *Atoms* describe synchronous change of state but do not ensure that it is coordinated. As such it can not be ensured that all readers of the state read the same value at the same time. However, the change is atomic; either old value or updated value are returned[15]. Atoms are computationally cheaper than Refs and are of use if coordination is of minor concern.
- *Agents* are similar to atoms as they do not consider the coordinated change. In difference to the former, however, change is processed asynchronously in shape of a function sent as a message which is eventually executed using the encapsulated agent state as parameter. The result is the new agent state.

Apart from the potential productivity advantage of the additional language paradigm its concurrency features introduce an alternative to the conventional locking mechanisms of Java. Considering the strength of the available agent concept, it merely classifies as asynchronous with regards to its execution; autonomy is of no concern (see documentation [Hic10b]).

[15]This satisfies the characteristic of consistency in transactions.

4.2.2 Android

The convergence of mobile and traditional desktop computing, indicated by the range of device types mediating this transition (e.g. net books, smart phones), is continuously taking place. Along with the increasing computing power available on mobile devices, operating systems and programming languages are beginning to converge similarly. One recent example of this development is the Android application platform which originally targeted the smart phone market but has yet grown beyond this scope and is increasingly considered as operating system for netbooks or the currently hyped 'pads' respectively tablet PCs.

Essentially Android is a software stack centered around a custom Linux kernel which integrates the actual hardware components (e.g. camera, wireless LAN, ...) and makes them accessible via an extensive libary set which includes network support, graphical user interfaces, security and various further functionality. Along with this Android comes with an own optimized virtual machine – similar to Java's one – which allows to develop applications using Java syntax. Atop of that Android provides a so-called application framework which is used to manage the different functional elements available in the system, such as Telephone Manager, Location Manager or Notification Manager. The use of the libraries along with those management elements are building blocks for own applications. The development principle is consistent – all provided applications (including the ones provided from phone vendors) are developed using the same mechanisms, thus allow the comprehensive change of all provided applications (e.g. the dialer application to make phone calls).

Although Android allows development using the Java syntax and provides a wide range of Java libraries, it falls short in providing the full set of functionality out of the box (e.g. Just-in-Time compilation, numerous libraries (e.g. Swing)). However its support goes further than the feature set of Java 2 MicroEdition (J2ME) (which does not provide reflection and serialization). A schematic overview of the full Android stack is provided on its website [Andd].

Apart from its architecture Android provides an innovative application development principle composing full applications from loosely coupled so-called application components which will be briefly introduced at this place – more information can be found on the according website [Andb].

Activities describe application components which run in the foreground, thus (typically) provide a GUI to the user and are usually rather short-running. Their execution (especially of inactive activities) is of lower priority; they will eventually automatically be removed from memory in case of lacking resources.

Services are the background complement of activities but are rather long-running. They are less likely to be subject of removal from memory.

Broadcast Receivers are a mechanism to capture raised events, be it system events (e.g. system started, SMS received) or user-defined events. They have a short life span and are just started in case of an event and stop execution once their code body is executed.

Content Providers encapsulate persistent unique storage locations such as the contacts list or the media library of the system and make those accessible to any other component.

The composition of those application component types into actual applications is realized by means of intents[16]. In the Android context, an *intent* "is a passive data structure holding an abstract description of an operation to be performed - or, often in the case of broadcasts, a description of something that has happened and is being announced." [Andc] Apart from this characteristic as an abstract message container – it holds both requests as well as general information – intents are processed asynchronously. Core properties of intents include actions to be performed (e.g. PICK, VIEW), context-related data in the shape of an Uniform Resource Identifier (URI) (e.g. tel://23729797) and categories, describing the desired capability of a target component (e.g. BROWSABLE (indicating that the intent (content) can be processed by/passed to an internet browser)). Additionally arbitrary custom information can be passed along in a HashMap-like data structure. Although the structure of intents is of generic type the dispatch mechanisms are

[16]In fact intents are only used to connect activities, services and broadcast receivers; the use of content providers is left out of focus at this point.

not; developers need to specify the target application component type for specific intents (e.g. Activity, Service) respectively sending mechanism (`startActivity()`, `startService()` or `sendBroadcast()`). The actual flexibility and expressiveness of intents becomes more apparent when introducing the notion of intent filters. *Intent filters* are attached to target components and describe their capabilities (i.e. actions, processable data and category). Based on those intent filters application developers can address application components either via explicit intents (which specify the target component's class name) or implicit intents in which case the target component is described using intent filters which are resolved to an actual target at runtime. Considering those features, intents allow a very loose coupling which eventually allows the exchange of components at runtime without necessarily breaking application functionality. Again to be mentioned, limitation to this loose coupling is that the application developer needs to know in advance which component type (i.e. activity or service, ...) to resolve. With these built-in asynchronous message passing and execution mechanisms, Android handles all concurrency aspects and keeps the application developer from dealing with those aspects – unless explicitly desired. This approach in fact allows a strong and flexible integration with components outside the actual application – especially when external functionality such as internet browser or phone application are used as those might differ in different environments, i.e. different phones. Eventually applications are only tied together by the application manifest, a XML file, which explicitly describes all application components as well as intent filters along with necessary permissions[17].

For clarification simple examples for explicit and implicit intents are given in Listing 4.1.

```
1  /* Creation of an explicit intent (for target ExampleActivity.class).
2   * The context from which intents should be sent is given
3   * (here: 'this' - the activity running this code) */
4  Intent intent = new Intent(this, ExampleActivity.class);
5  startActivity(intent);
6
7  /* Creation of an implicit intent.
```

[17]Android applications need to have explicit permissions to perform tasks such as accessing contacts or internet resources.

```
8     * It asks the user (via GUI) to pick a person from the
9     * contacts list (activity to do this is not given)
10    * and returns the result (via another intent) */
11   Intent intent = new Intent();
12   intent.setAction("ACTION_PICK");
13   intent.setData("content://contacts/people/");
14   /* Limitation: The component target type needs to be known (here: activity). */
15   startActivity(intent);
```

Listing 4.1: Examples for Android intents

Summing up, Android provides a mechanism combining both asynchronous message passing (in contrast to Clojure's STM) and loose coupling for the construction of applications in a flexible manner. Its infrastructure in fact bears similarities with the one of multi-agent systems. This aspect motivates its consideration for this work and is explored at a later point.

4.3 Java-based Asynchronous Message Passing Frameworks

The concurrency properties of the technologies discussed above take a clear (and certainly opposing) position on how to model communication in concurrent systems. Still, those technologies target specific areas – Clojure as a LISP dialect for interactive programming and Android's mechanism runs only on the operating system itself. In order to show the landscape of Java-based asynchronous message passing frameworks and to select candidates for direct integration into a multi-agent platform a performance benchmark has been undertaken to show specific strengths and weaknesses. The results of this investigation will be briefly replicated at this place. For a more intensive discussion and introduction of the frameworks refer to [FNP10].

Although earlier benchmarks with regards to the performance of message passing frameworks have been undertaken (e.g. by Karmani et al. [KSA09]), they particularly focus on an actor-centric view – and thus design according scenarios. While one objective of interest, performance, is similar in this context, the scenario is done in a more agent-like fashion and considers a differing set of message passing frameworks:

This scenario involves a parameterized number of agents allocated on a two-dimensional grid. Every agent initially holds its coordinate as a state, ensuring a unique state for all agents. A subset of all agents is then activated. Those request 'state' from other randomly chosen agents. If not in a transaction with other agents, agents will accept those requests and subsequently exchange the state, i.e. the coordinate. Along with the state exchange the activity is passed along; agents which have successfully requested a state exchange become inactive and their counterparts actively request the next exchange with a new randomly selected partner. This procedure is repeated until the first agent succeeded in exchanging state for a parameterized number of times, in this case 1000 times. The benchmark stops timing and collects the number of all transactions to calculate the message throughput of the framework. Along with this the standard deviation of achieved transactions of all agents is calculated to determine the fairness with which agents have been activated to exchange their values. In order to make the frameworks comparable, a unified HashMap-based message structure has been introduced where feasible.

The results of the benchmark, reproduced in Figure 4.2 (performance) and Table 4.1 (fairness), reveal various performance and fairness clusters which are related to messaging principles and internal architecture.

Number of agents	Kilim	Jetlang	Actor Foundry	Actors Guild	Korus
100	25.1	465	431	425	25.5
225	25.4	446	432	433	24.3
10000	24.7	431	432	–	26.4
40000	25.7	232	232	–	27
160000	25.7	115.8	115.8	–	27
250000	25.7	92.67	96	–	27

TABLE 4.1: Fairness of Message Passing Frameworks (as standard deviation of rounds) for selected number of agents

Those results will be summarized and briefly interpreted at this point. Context-relevant information about the frameworks is provided.

The most significant difference in performance results can be perceived for the messaging frameworks which provide safe messaging by serialization – and as

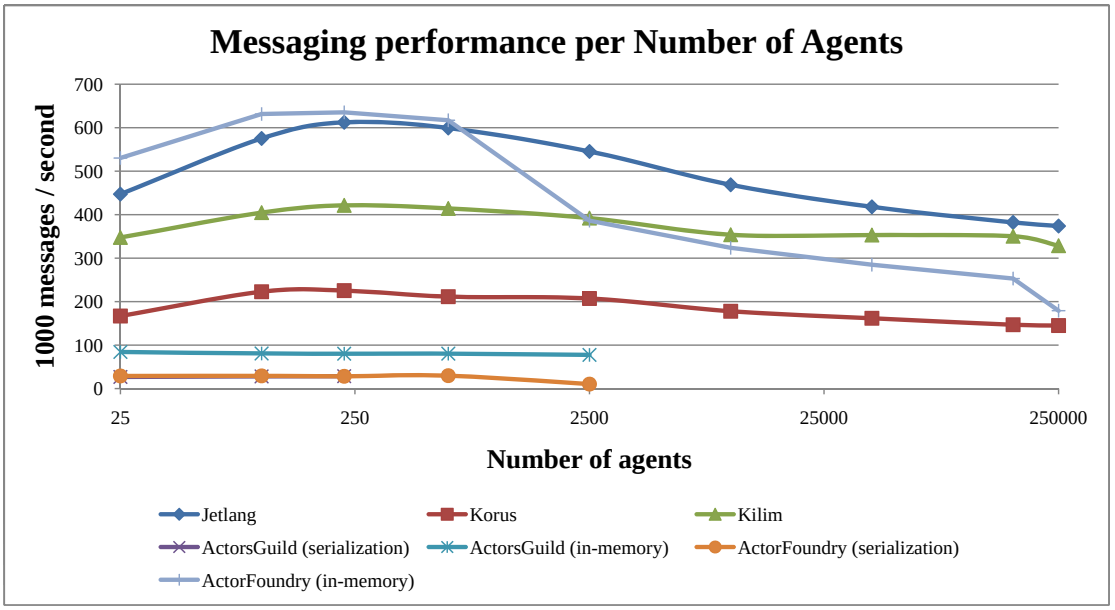


FIGURE 4.2: Performance results for Asynchronous Message Passing frameworks

such satisfy one key criterion of the requirements for actor frameworks as defined by Agha [Agh86] and discussed in subsection 4.1.2. Only two of the surveyed frameworks provide support for serialization during transmission, namely ActorsGuild [Actb] and ActorFoundry [Acta]. Eventually their throughput is equally low as the Java serialization is the bottleneck during processing (see the lowest (overlapping) curves in Figure 4.2). The serialization approach is prohibitive for high-performance message passing (especially when considering the micro-agent concept). In contrast, the use of in-memory message passing, i.e. the sending of pointers to messages is significantly faster but bears the risk of manipulation of actual message objects after sending. However, for the context of this work this risk is acceptable.

For the in-memory versions of the frameworks (both frameworks mentioned above provide a serialization mechanism as well as passing of memory references) two fast ones, namely ActorFoundry and Jetlang, can be identified. However, ActorFoundry's performance continuously declines throughout the test, indicating a limited scalability. Jetlang remains the overall fastest framework. The alternative Kilim [SM08] has a balanced performance eventually achieving similar throughput as Jetlang for higher number of agents. A framework constantly ranging at

around half the throughput of Kilim is Korus [Kor]. ActorsGuild shows the weakest performance for in-memory message passing – and eventually quits execution for larger number of agents.

Considering the fairness results, clusters are clear-cut. Kilim and Korus show similar fairness values ranging within the unfairness caused by Java's pseudo-random number generator. All other frameworks tend to have a considerably unfair allocation of resources, all of them showing standard deviations of more than 400 transactions. Given this background, the framework which is most successful in managing the trade-off of being both fast and fair is in fact Kilim. However, the reason for its performance is hidden in its architecture. It provides a centralized scheduler with according worker threads backing the individual communicating entities. In order to achieve a fast dispatch of messages it introduces the post-compilation step of 'weaving' – a reference to the framework's name – which operates directly on Java bytecode and eventually resolves the 'method yielding' in Java class files with explicit references (see Srinivasan and Mycroft [SM08] for details). A framework showing similar architectural principles but avoiding this proprietary change of class files is Korus. An example for a framework not providing a central scheduler but delivering messages in a best-effort manner between different thread-backed *fibers* is Jetlang. Although the performance is high as of the lacking central scheduler, the fairness is limited as no central instance controls a fair dispatch of messages.

Summing up, asynchronous message passing frameworks for Java are considerably widely available but – although calling themselves actor frameworks – give up safe messaging characteristics in favour of fast passing of memory references. This bears the risk of manipulation by the sender after sending but is yet the sole approach to allow high-performance asynchronous message passing in Java.

Given the introduction to concurrency handling mechanisms in different technologies (Clojure and Android) as well as native Java, we now have a basis for design considerations for the reimplementation of the micro-agent framework.

Chapter 5

Reimplementation of the Micro-agent concept

After covering the base concepts relevant for this work and the research fields surrounding it, now the actual agent framework of concern, OPAL – and specifically its micro-agent layer –, is introduced. After introducing its concept the limitations of the current implementation are highlighted. As a result amendments to the actual concept as well as concrete requirements for its redesign are identified. Following this the actual design with consideration of the introduced technologies is described.

5.1 Existing Micro-agent Framework and Requirements for a Successor

5.1.1 The existing concept and implementation

The Java-based Otago Agent Platform (OPAL) [NBPC01] is a general-purpose agent platform. Its development started nearly one decade ago in the context of the New Zealand Distributed Information Systems (NZDIS) project at the University of Otago. OPAL itself is an implementation of the FIPA Abstract Architecture (see subsection 2.2.3) and as such satisfies all necessary requirements by providing components such as Agent Management System (AMS), Directory Facilitator (DF) and Message Transport System (MTS) – AMS and DF are implemented as agents themselves. Agents developed with OPAL are FIPA-compliant with respect

to architecture and agent communication language. As OPAL is completely written in Java and – on this high level – driven by standard-compliance, the MTS is an implementation of an extended version of the Java Agent Services (JAS). JAS is effectively a set of Java interfaces and is specified via the Java Community Process[1] to allow a standardized mapping of (technology-independent) agent communication onto specific network level transport services – supporting the idea of 'pluggable' services.

Internally OPAL presents itself in a less standardized manner: The rather coarse-grained OPAL agents are extensions of so-called *micro-agents* which represent " ... the lowest and most primitive level of agent instantiation." [NBPC01] Micro-agents trade commitment to FIPA standards against flexibility and performance and are the key to OPAL's flexible multi-level approach to AOSE.

The micro-agent concept, originally introduced as KEA[2] [NPC01], describes an organisational MAS model – putting emphasis on the agent organisation and the decomposition aspects rather than collections of individual agents. As such the *agent* understanding is essentially of weak nature and is defined as a persistent entity in a MAS playing " ... one or more particular roles in a society of agents" [NPC01]. *Roles* in turnaround are specificiations of "cohesive sets of behaviours, functions or services in a multi-agent society" [NPC01]. Role specifications are independent from the implementation, i.e. each agents can use varying implementations to satisfy the role specification. Micro-agents themselves can be *responsive*, i.e. do not control an own thread of execution but are reactive towards external stimuli. Once activated they can initiate social interactions themselves, including the ability to react towards (execution) goal requests (e.g. by means of commitment, refusal or fulfillment). *Autonomous micro-agents* control their own thread of execution and can manage a group. A *Group* is modelled as a role specialization. The agent playing this role manages the according group; groups only have one leader, the group owner. Agents can be functionally decomposed into sub-agents (which are micro-agents themselves). The management features (in-group communication,

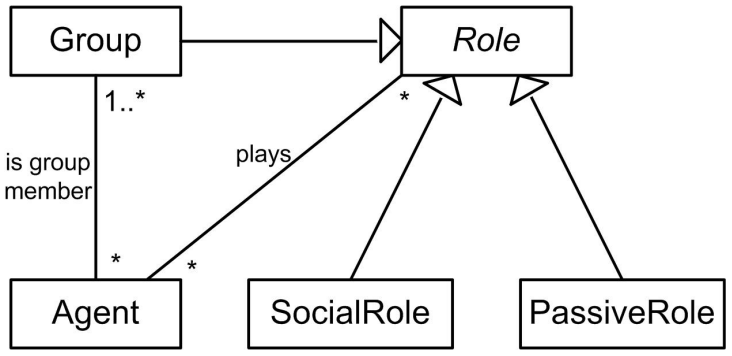
[1] It is specified in Java Specification Request 87 [JSR] but has never found introduction into any Java specification. Its sources can today be found under [JASb].

[2] KEA is a recursive acronym for 'KEA Enterprise Agents'.

life-cycle management) of the group mechanism allow this to take place in a structured manner. If agents are composed from sub-agents (and in fact show direct or indirect cooperation behaviour) the composed higher-level agent can be considered an *agent system*.

The concept recognizes agents and roles as first-order concepts but from a software engineering point of view more emphasis is put on the role – in contrast to the agent which is seen from an infrastructural point of view: Application developers extend (specialized) roles whose instances are attached to an agent which merely serves as runtime container. In turnaround agents in the micro-agent package do not exist without at least one instantiated role; agents die when giving up roles, yet only can add roles, not remove those (in case of multiple roles). This differs from the actual OPAL agents which extend the default agent implementation but provide more explicit role handling capabilities to allow the developer to handle the registration of roles (not all roles played by an agent should necessarily be announced in the context of open systems) with the FIPA-specified discovery mechanisms. Role information of micro-agents is generally accessible for discovery by other (and only) micro-agents. Figure 5.1 provides a brief overview of the high-level concepts in the original concept[3].

Considering the (performance-) efficiency-driven simplistic notion of this concept



Visualized following: Nowostawski et al. [NBPC01]

FIGURE 5.1: Original Micro-agent Concept

(from hereon summarized as *efficiency principle*) and low level of abstraction from

[3]A similar overview is documented in the original publication [NBPC01]; the figure provided here considers terminological changes in the reimplementation (which do not have semantic implications).

implementation, its attractiveness becomes clearer when looking at some of its fundamental mechanisms. Those are the agent organisation, communication and means of agent discovery.

All agents on the system are implicitly forced into a hierarchical[4] relationship via the aforementioned group concept and belong to at least one group whose owner can in fact control its group members. At the highest level a special agent implementation, the so-called SystemOwner, is introduced to satisfy this condition for every level – it is the only agent being in a recursive relationship as being its owning group's owner and does not provide any further capabilities. This principle enables a consistent agent/role discovery mechanism throughout all levels of agent systems.

Communication takes place by means of method calls. Roles need to implement the Provider and/or Customer interface which enforce the implementation of performative-named methods (such as `achieved(Provider sender, Goal goal)`) which in turnaround include the role-specific implementation for goal handling. The necessity to provide the calling role as a parameter is based on the Java limitation that the origin of a method call can not be identified by the callee. The requirement to implement the according interfaces (i.e. `SocialRole`) provides the key for role-based discovery of agents in the platform. As such the micro-agent platform strongly relies on Java's dynamic proxy mechanism which is part of the Java's reflective meta-programming capabilities. This is similarly the case for goals which implement a Goal interface which is the basis for arbitrary (user-defined) specializations of goals. Goal objects encapsulate all necessary information for their execution without a unified structure, reducing the necessity to provide further meta-data. Both sender and goal executer need to interpret specific goal specializations which can be distinguished via the Java class hierarchy. A collection of pseudo-code examples shall be given to clarify the explanation given to this point.

[4]Being aware of the etymological inadequacy the use of *levels of agent systems* represents an alternative.

Listing 5.1 shows an example implementation for a service provider which implements the according Provider interface and its `request()` method[5].

```
1  public class ServiceProvider implements Provider {
2
3    public void request(Customer sender, Goal goal){
4      if(goal.getClass().equals(ServiceGoal.class)){
5        sender.commit(this, goal);
6
7        /* process goal and respond with sender.achieved() or
8           sender.failed(), .... */
9
10     } else {
11       //Goal received is unknown
12       sender.unknown(this, goal);
13     }
14   }
15   //further method implementations (e.g. subscribe()) are omitted
16 }
```

Listing 5.1: Implementation of ServiceProvider

Listing 5.2 shows the according customer counterpart implementing the performative methods for potential replies of the provider as well as a custom `startInteraction()` method which initiates the interaction between customer and provider.

```
1  public class ServiceCustomer implements Customer {
2
3    public init(){}
4
5    //implementation of interface method
6    public void commit(Provider sender, Goal goal){
7      //process commit of provider
8    }
9    //implementation of interface method
10   public void achieved(Provider sender, Goal goal){
11     if(goal.getClass().equals(ServiceGoal.class){
12       System.out.println("Goal result: " + goal.toString());
13     }
14   }
15
16   //further implementations of interface methods are omitted
17
18
```

[5]The remaining methods of the interface are omitted as of lacking relevance for this example.

```
19    //helper method defined to initiate interaction
20    public void startInteraction(){
21      //the goal to be processed
22      ServiceGoal goal = new ServiceGoal();
23      //finding and binding of available roles to proc
24      ((ServiceProvider)SystemAgentLoader
25            .findRoles(ServiceProvider.class)[0]).request(this, goal);
26    }
27  }
```

Listing 5.2: Implementation of ServiceCustomer

The main method shown in Listing 5.3 ties the example together by instantiating the according role implementations and loading them via the so-called SystemAgentLoader which instantiates according agents for the respective roles.

```
1  public static void main(String[] args){
2    //initialize agent with according role
3    SystemAgentLoader.newAgent(new ServiceProvider());
4    ServiceCustomer customer = new ServiceCustomer();
5    SystemAgentLoader.newAgent(customer);
6    customer.startInteraction();
7  }
```

Listing 5.3: Main method to start interaction

Upon initialization the `startInteraction()` method on the ServiceCustomer is called[6]. The `startInteraction()` method makes use of the discovery mechanism respectively the dynamic linking functionality (see lines 24/25 in Listing 5.2). `SystemAgentLoader.findRoles()` is one of various available methods to retrieve all roles of a given class instantiated in the agent framework and returns those in shape of an array. The user is free to evaluate the result and choose any returned role (in our case the first value of the array (as there will be only one)) but needs to cast it to the according social role specialization (as values are returned as a Role array as the user may want to lookup roles other than social roles) in order to invoke the according method (here `request()`). The `request()` method will be invoked on the according agent (see line 3 in Listing 5.1). The goal passed via the

[6]Here the explicit reference to the role instance and call of the `startInteraction()` method on the role instance (see line 4 and 6 in Listing 5.3) indicates the role-centrism of the agent framework.

method call is checked for its type before proceeding with processing. To do this a call to the reflective `getClass()` method ensures the proper goal specialization (see line 4 in Listing 5.1). In the positive case the ServiceProvider calls the `commit()` method on the Customer interface; in the negative case he calls the `unknown()` method which represents the equivalent to FIPA's NOT-UNDERSTOOD performative. Upon successful processing the customer's `achieved()` method (see line 10 in Listing 5.2) is invoked using its reference (sender parameter) passed along with the initiating request.

This short description of the interaction[7] shows the low level of implementation of micro-agents. Their implementation hardly abstracts from its underlying implementation language; the KEA micro-agent framework uses pure language capabilities which – depending on the reader's standpoint – let agents appear as simple as objects. However, the dynamic linking mechanism enforces a very loose coupling and robust execution; agents do not 'know' the service-providing agents which will eventually 'fulfill' their goals.

In consequence a simple interaction makes at least two method calls necessary to complete a transaction. In order to further reduce potential performance penalties the most simplistic role type can be of arbitrary type (as long as it implements the Role interface). Roles of this kind can implement arbitrary individual methods (e.g. `talk()`) (as long as they allow external invocation (using the `public` modifier)) which potentially return values. They can be discovered using the same mechanism as for social roles. But given this specific implementation the coupling can be considered less loose as the caller needs to have specific knowledge about the callee (i.e. its method signature).

From a software engineering point of view it is clear that, albeit being hardly distinguishable from objects, the micro-agents enable the application developer to consistently maintain an agent-oriented modelling perspective up to the atomic level without sacrificing performance by additional abstraction.

Regarding the essential AOSE characteristics (Decomposition, Abstraction and

[7]Here a sequence diagram showing the interaction might have been useful. However, in this context focus was put on implementation stubs to clarify the tight linking of the existing micro-agent platform to the Java programming language – using features hardly to be found in other (non-Java or non-JVM-based) languages.

Organisation), the implicit group management and enforcement of an organisational relation ensure the consistent structure of developed applications. Decomposition of functionality into different agents is often a simple concern, particularly for developers from the OO field. Implementing abstraction by means of agents, however, can remain hard, especially when considering strong narrow platforms; their support for organisations is rather flat. Micro-agents however, offer the alternative approach to model abstraction by means of sub-agents which does not only allow intuitive modelling but also to retrace it in developed applications as of the explicit representation in the code. This eases the understanding of complex applications and allows to retrace earlier application design decisions. A very simplistic example (with the artificial CustomerSubAgentRole() which can be any user-defined role implementation) for the initialization of sub-agents is shown in Listing 5.4.

```
1  public static void main(String[] args){
2    //instantiation of role
3    ServiceCustomer customer = new ServiceCustomer();
4    /* initialization of agent to play the role. Agents without explicit group
5     * association (loaded by SystemAgentLoader) are sub-agents of top-level
6     * agent (SystemOwner). */
7    AgentController control = SystemAgentLoader.newAgent(customer);
8    //instantiation of sub-agent role
9    CustomerSubAgentRole subagent = new CustomerSubAgentRole();
10   /* first the group of the started agent is called,
11    * then a new agent (for the sub-agent role) is initialized within that group */
12   control.getGroup().getAgentLoader().newAgent(subagent);
13 }
```

Listing 5.4: Example for sub-agent initialization

Figure 5.2 visualizes this pragmatic approach of micro-agents (or μ-agents) towards AOSE and makes the application of the core AOSE modelling objectives 'Decomposition', 'Abstraction' as well as the inherent 'Organisation' in the shape of 'levels of agent systems' explicit.

5.1.2 Limitations and Requirements for the Successor

The very simplistic and efficiency-motivated (which is yet to be shown) design of the current concept allows a consequently agent-oriented modelling but has some practical limitations:

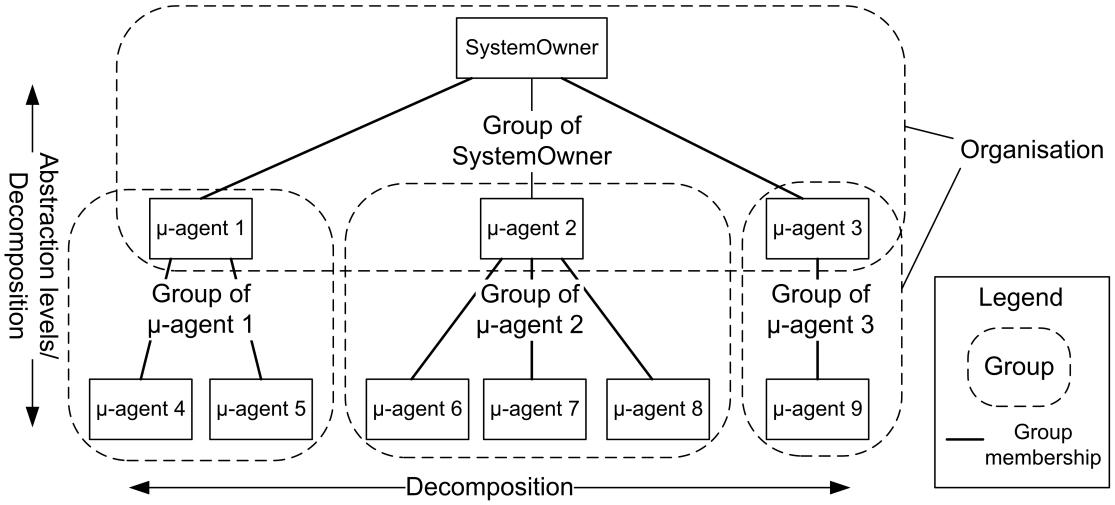


FIGURE 5.2: Representation of AOSE properties in μ^2

In the default implementation threads are hardly considered – the method-based interaction is effectively purely sequential which can be well-performing for simple tasks which merely exploit the loose coupling. Depending on the view the micro-agent implementation does not actually allow the modelling of agent-based applications as it does not support asynchronous concurrent conversations between agents.

The dynamic binding is a considerably explicit task for the developer (who actually models the binding himself) and involves significant understanding of the framework internals. Along with this the mechanism simply involves a considerable amount of code (see line 24/25 in Listing 5.2).

Additionally the framework has mechanisms to control agents (such as the `AgentController`) but does not involve systematic platform management (e.g. on application shutdown). This can be potentially harmful when considering the use of agents for I/O activities in which case according resources should be handled properly before shutting down the application.

Another issue lies in the role-centricism of the concept. Interaction is modelled between roles, not necessary agents. In consequence the lifetime of an agent is ultimately tied to the roles it plays. If roles are disposed, the agent dies. Although this is acceptable in the original concept as the creation of new agents is considered cheap, it is ineffective when considering changing role compositions of agents over time (as roles cannot be disposed but only added) and in the context of

longer agent runtimes in general. Effective productivity advantage (and eventually performance advantage) could be gained by taking a more agent-centric approach while changing roles on a more frequent basis.

Further the strict enforcement of the explicitly hierarchical regime (every agent is forced into this organisation pattern) which – depending on the application context – might not be adequate. An example for the latter is the use of social simulations where agents controlling a scenario (or representing the environment) should not ultimately be included in direct agent interaction (respectively not be directly visible/discoverable as an actual agent). This would harm the provision for a controlled environment (in MABS) in contrast to open systems (in general purpose MAS).

Along with the enforcement of hierarchical relations[8] it needs to be mentioned that the understanding of sub-agents as subsystems (which should eventually be the case) is not consistent with Bunge's system understanding: Even when being sub-agents micro-agents can still directly interact with agents outside their group – breaking with the idea to delegate this functionality to the higher system level, as in MadKit. This direct interaction is considered relevant to satisfy the necessary performance criterion by weakening those rules. In addition to this an alternative approach should be provided which allows communication consistent with the system-theoretical foundations introduced earlier (see subsection 2.2.1).

Many of the deficiencies mentioned before are of minor concern when using the micro-agent framework in conjunction with OPAL which adds the necessary platform features. But in order to make the system fully self-sustained and envision it as a high-performance alternative agent life-cycle aspects need to be modelled comprehensively and global management features integrated, making a transition from an agent framework to a fully self-contained agent platform[9]. Finally a feature relevant for agent-based system is the potential distributedness. In order to *externalize* the advantage of the high performance of micro-agents it is useful to

[8]To recall: Micro-agents can be part of multiple groups but will always have a primary group relation.

[9]However, in the rest of this text 'framework' and 'platform' describe the same concept. The concept of 'framework' puts a stronger emphasis on the extensibility and agent implementation functionality while 'platform' indicates a wider scope, especially respecting the management aspects of the system.

provide access to network resources respectively allow multiple nodes' micro-agents to interact. This should especially be seen from the context that micro-agents are interaction-centric units – in contrast to high-level (and more ego-centric) OPAL agents. As such efficient communication in general (and as such including the network) should be of stronger concern – even if it does not comply with the FIPA specifications[10].

Given the limitations of the current platform as guideline along with the consideration of additional functionality and integration of new technologies a set of requirements can be specified for a succeeding micro-agent package. The listing of requirements will not be structured by non-/functional requirements but rather apply the notions of *behavioural requirements* and *developmental quality attributes* (as suggested by Clements (and documented by Faulk [Fau97])) as of the more intuitive fitting of requirements.

Behavioural requirements are thus:

- Handling of *threads* – Application developers (= framework users) should not need to deal with threading concerns – unless they intend to do so – and assume that the framework is inherently concurrent.
- Handling of *dynamic binding by the framework* – All dynamic binding aspects should be fully handled by the framework to allow a consequently loose coupling of agents. Yet the user has to realize this explicitly.
- Satisfaction of the *efficiency principle* of micro-agents in relation to the original framework and other existing platforms is of importance. A certain performance loss – especially for sequential operations – needs to be anticipated when introducing an abstraction layer for asynchronous message passing. However, performance should remain one key advantage in comparison to other frameworks.[11]
- Communication

[10] FIPA only standardizes external communication of agents; the choice internal communication mechanisms is left to the platform implementations.

[11] This requirement is particularly interesting in the context of the 'behavioural requirements' as it is a typical example for an item of the category of non-functional requirements.

  - Extended *support for communication patterns* – communication is the strength of micro-agents (not necessarily knowledge-level communication using ACLs); the framework should provide powerful support to developers.
  - Communication should be done using a *unified (but extensible) message container*. This allows better transparency (e.g. debugging, logging) and stronger abstraction of message content from communication vehicle.
  - Provision of *network communication* – Micro-agents shall be able to communicate efficiently between different network nodes.
  - Communication options consistent with systems theory – Modelling mechanisms which are consistent with system-theoretical level understanding shall be suggested.

- Consistent handling of the *Agent organisation* – Agents should be consistently managed upon eventual shutdown (e.g. coordinated shutdown of all agents upon system event) but also consider options to exclude agents from the enforced organisational integration.
- *Loose coupling between agent and role* – Roles and agents shall be less coupled to provide more flexibility for the application developer with regards to flexible role assignments (e.g. life-cycle independence of agent from role).
- *Integration of Clojure* – Clojure shall be provided as an alternative functionally-inspired agent implementation language (along with Java).
- Agent-based Simulation
  - *Fair scheduling* – Fair scheduling of agent behaviour is crucial for the construction of replicable, and overall valid simulations.
  - *Data collection and reporting* mechanisms shall be considered as of their relevance in the context of social simulation.
- A *port for the Android OS* shall be provided with the target to achieve application code compatibility and network interoperability.
- The reimplementation should maintain *compatibility with OPAL platform* as a whole respectively allow the reintegration.

Developmental quality attributes include:

- The resulting system should have a minimal memory footprint.
- The extensibility of the resulting software needs to be ensured.
- The software needs to be testable for future extensions.
- Compatibility to the lowest possible Java specification (e.g. 1.5 or lower) should be maintained.
- Third party software integrated should be available under open source licenses to allow the platform itself to be distributed as open source.

The listing of the requirements includes a prioritization of the items mentioned on top of the list.

Given the original concept and its limitations along with newly derived requirements a design is suggested in the upcoming section.

5.2 Design and Implementation of the Micro-agent Platform μ^2

As a first step the requirements are separated by work packages which are loosely structured by the areas micro-agent management (agent-centric reimplementation, platform management features), communication (local message passing, network communication), simulation-related extensions (Clojure, simulation-related extensions (scheduler, reporting)) and development of the mobile version (portation to Android). The areas of micro-agent management and communication are ultimately linked and define the core of the revised micro-agent framework. This decision is taken from an infrastructural point of view, focusing on the efficiency principle of interaction and the provision of a concise core framework. Further functionality should be – where applicable – added in a modular fashion.

5.2.1 Design

The actual design process enforces frequent review and thus takes place in an iterative manner rather than a linear development approach (with strict separation of design and implementation stage as in the traditional waterfall model). Key reasons for this are the performance constraints set up in the requirements which demands for a careful balance to manage the trade-off of providing convenient

abstractions from the actual communication mechanisms and performance.[12] The resulting design for the core of the framework comprises of three layers and is described in the following and schematically visualized in Figure 5.3.[13] The top layer, the *Agent Logic Layer* (ALL), represents the elements from the original architectural metamodel such as agent, role and groups. This design suggests some changes to this model. The original model is considerably role-centric (see subsection 5.1.2). To provide the developer with more flexibility agents shall optionally be initialized without role but may add roles and remove roles at runtime (the latter is not possible in the original concept without destroying the agent). Nevertheless the default principle will remain to start an agent with some role behaviour. Minor side aspect is the mere change of the 'goal' term to 'intent' as this represents the abstract nature of agent requests without confusing those with the goal term and the conceptual implications surrounding it. Goals are hardly explicitly communicated[14] (but are represented internally) and the extended understanding developed in the context of further research, such as various goal specializations (e.g. maintenance goals) which do not conceptually match the understanding in the micro-agent platform; Intent(ion)s are thus used to represent abstract requests externally.

Along with those conceptual changes the platform now strictly manages agents upon shutdown of the platform to allow the implementation of long-running agents with changing capabilities as well as to avoid memory leaks (by unclean termination of agents (and release of resources) in case of repeated platform start and shutdown (without unloading the JVM) such as during tests). In consequence the static structure of the original micro-agent concept remains fairly similar – changes include the coupling of concepts and their runtime behaviour.

The switch from a role-centric to a more agent-centric approach is also necessary to allow a clear addressing scheme for individual agents. Agents are now consequently named (be it user-defined or automatically). This in turnaround puts

[12] In order to allow the measurement of the performance a simple benchmark whose scenario had been initially suggested in [NPC01] is used. It is described when discussing the implementation outcome.

[13] The message flow for agent activity within the platform is indicated by numbers.

[14] In fact micro-agents do not have the capability to 'share their goals' as no common knowledge representations such as ontologies are available on this level.

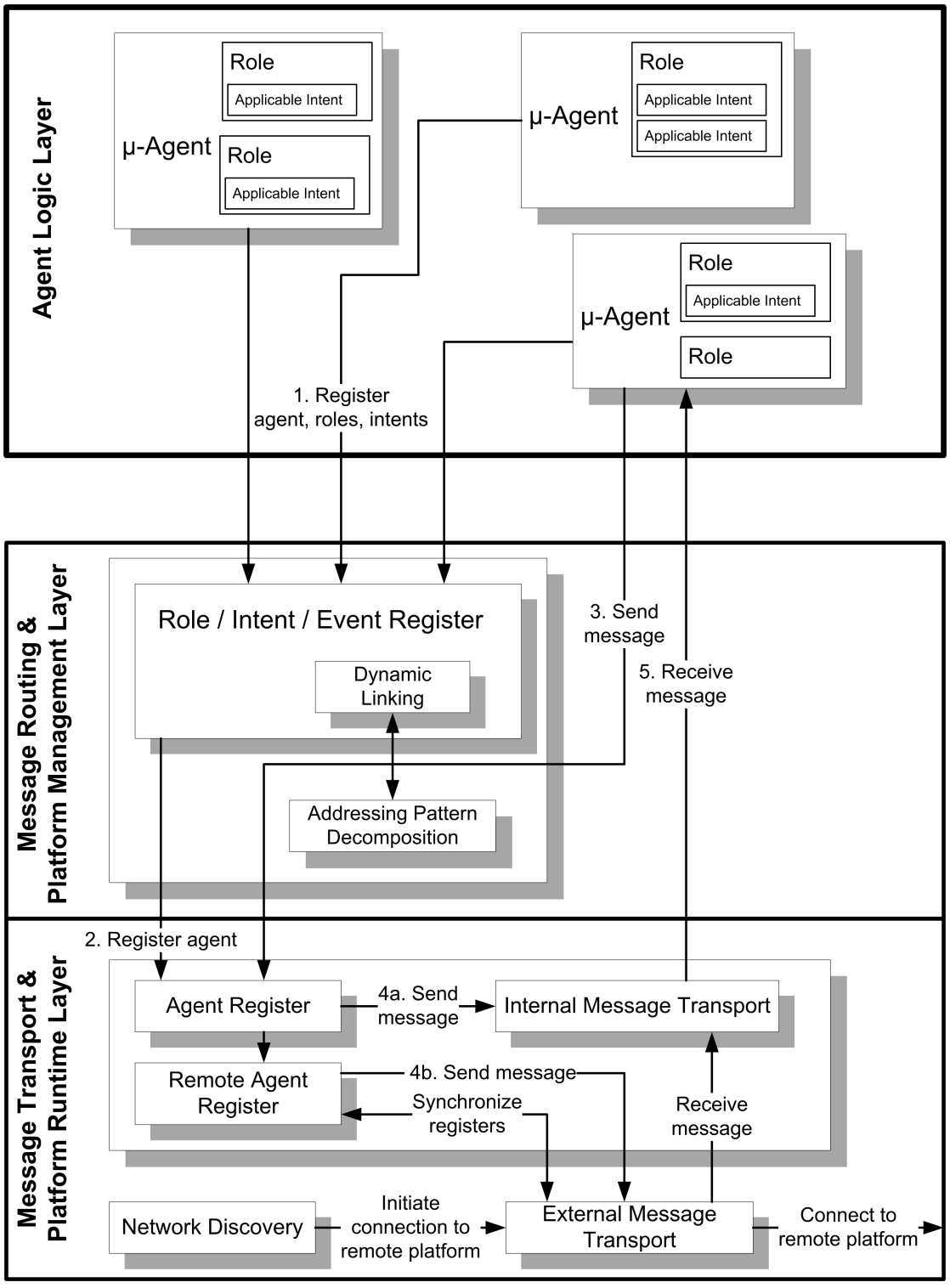


FIGURE 5.3: Platform layers for micro-agent platform μ^2

Addressing pattern	*Description*
Unicast	sends message to one specified recipient
Broadcast	sends message to all registered agents (differentiation between local and global (network-wide) broadcast)
Multicast	sends message to event subscribers, thus modelled via event subscription mechanism
Groupcast	sends message to all agents which are members of a certain (agent) group
Rolecast	sends message to all agents playing a given role
Randomcast	sends message to a specified (or random) number of randomly chosen agents
Customcast	sends message to agents specified by the combination of various aforementioned patterns

TABLE 5.1: Addressing patterns for agent communication in μ^2

higher demands on the agents themselves as they now need to incorporate simple reflective mechanisms with regards to their roles' capabilities (i.e. 'know' their applicable intents). This is necessary to identify the appropriate role for the delivery of requests received by an agent. In the original framework this is not of concern as roles directly invoke each other. Naming schemes are further introduced for roles as well as groups to ease differentiation in a mnemonic fashion.

In addition to this the new concept will include an extension of addressing patterns. Yet agents interact in a unicast fashion which limits modelling of complex interactions. The addressing patterns listed in Table 5.1 are introduced in the new platform.

All of the addressing mechanisms work in a distributed fashion, thus show effect across different (connected) platform nodes. Only the Groupcast receives special consideration as group members residing on different nodes can be addressed using the group owner's name. Groups themselves however, cannot be spread among different nodes as groups play a central role in platform management. Eventual network connection loss could thus leave to 'loose' agents which is avoided by unambigious assignment of agents to one local group, defining the *primary group relation* (in the default case the SystemOwner's group) via which agents receive management commands. Apart from this agents can join arbitrary further groups if useful for a particular application.

The use of those predefined addressing pattern provides the application developer

with a powerful tool anticipating various interaction patterns.[15] The combination of those built-in operations additionally allows the modelling of more complex communication patterns (especially via Customcast (Example: send message to fixed set of agents along with two randomly chosen ones and one group while excluding specified agents)). The patterns described here are of particular relevance for the application developer – and thus the agent logic layer.

In order to provide a mechanism to ease the decomposition in the platform and allow modelling in consistence with the systems theory, the concept of message filters is suggested as a specialization of communicating roles. Although of strong design implication, the understanding will be more intuitive in conjunction with related implementation considerations (as of the interdependency). In consequence message filters are described in subsection 5.2.2.

The lower layers which build the infrastructural backbone are newly introduced as part of the redesign. As the earlier framework is fairly self-contained based on its method-based interaction, the successor is backed with explicit message passing mechanisms, which makes the introduction of interfaces to a transport level necessary. In consequence the Agent Logic Layer merely maintains basic message delivery semantics for individual agents (e.g. dispatching received message to according role on agent).

The *Message Routing & Platform Management Layer* (shorter: Message Routing Layer (MRL)) serves as an intermediate layer holding information on agents' roles, their applicable intents, manages event subscriptions and preprocesses messages sent by agents to decompose the different addressing patterns. The registry management on this level (see Figure 5.3) is concerned with capabilities of individual agents[16]. The register is used to route messages according to the requested addressing patterns (see Table 5.1) respectively to establish the actual intent- or event-based discovery of recipients (dynamic binding). This layer decides on the actual routing of outbound messages and decomposes addressing patterns (into

[15]Especially the Randomcast is of relevance when considering social simulation applications with randomly selected recipients. It could also be used to model the notion of an Anycast (as specified in the IPv6 specification).

[16]With regards to the separation of registered capabilities from actual agent register this loosely reflects the functionality of a yellow-page registry (as the Directory Facilitator in the case of FIPA-compliant platforms).

either Unicast or Broadcast delivery and dispatch to local or remote agents)[17]. To do this it accesses the actual agent directory (comprising of both local and remote agents) held on the lower Message Transport & Platform Runtime Layer and combines it with its role and intent meta-information. This latter clarifies the intermediacy of the Message Routing Layer and additionally leads to the decision to delegate the control mechanisms for the entire platform to this level as it can both access the agent organisation on ALL level as well as the lower Message Transport Layer.

The *Message Transport & Platform Runtime Layer* (shorter: Message Transport Layer (MTL)) ultimately holds the agent directory (which associates agent name with address) and is based on agent registrations passed through by the upper layer (which filters role and intent registrations). The agent directory is held on this level in order to map agent identifiers onto the according transport mechanisms in a transparent manner and to avoid a performance penalty by accessing higher layers. As this performance-sensitive area is likely to receive frequent improvements the coupling to an underlying message transport framework shall be kept as limited as acceptable in the context of the efficiency principle.

This layer also incorporates network-related functionality. Apart from managing the network communication a network propagation mechanism is suggested to maintain a network-wide synchronized agent directory to reduce lookup times for remote agents. As this directory is ultimately linked to network connectivity the information is maintained on this layer. Last (and probably least relevant but very convenient) network feature is an automatic discovery of other platforms instances running on the same network. With this functionality application developers (and ultimately application users) can build distributed systems without the need to care for the network-level (i.e. ip addresses, hostnames). In consequence the platform provides event-based notification on network-level state changes (e.g. connection of new platform) and is integrated with the platform-wide event subscription mechanism (which allows agents to subscribe to those system-level events).

[17]Motivation for this decomposition is to limit the requirements for message passing frameworks on the transport level (as those do not generally support the partially sophisticated addressing patterns out of the box).

Concurrency handling mechanisms respectively threads are concentrated on this lowest level of the platform, the system kernel, in order to offer the highest possible abstraction from those mechanisms to the agent layer (ALL). However, actual platform management (e.g. shutdown) is realized via the management layer (MRL). Apart from configuration aspects platform developers are thus only concerned with the higher layers, the Agent Logic Layer and Message Routing & Platform Management Layer.

5.2.2 Implementation

With the high-level concept of the new micro-agent platform in mind, the following discussion of selected implementation aspects clarifies the realization of those design decisions.[18]

The designed platform layers are separated into two (Java) namespaces. The two upper layers (*Agent Logic Layer* and *Message Routing & Platform Management Layer*) – which are complementary with regards to the representation of the full agent semantics and concepts (i.e. roles, intents, events) – are combined into one namespace (*org.nzdis.micro*). The lower – and presumably more prone to future technological changes – message transport-intensive layer is separated into the namespace *org.nzdis.micro.messaging*.

Implementation of the Agent Logic Layer

On the highest layer various interfaces from the original KEA framework have been reused and extended to maintain a reasonable degree of compatibility to the OPAL platform. Types of micro-agents are differentiated by their role implementation. *SocialRoles* make use of the integrated message passing capabilities while *PassiveRoles*[19] only allow simple invocation using the pattern described for the KEA framework without assuming a fixed role signature (i.e. no fixed method interface (e.g. `request()`, `subscribe()`, ...) is assumed). The newly introduced functionality as described in the concept will mostly only apply for SocialRoles as it relies on the message passing facilities which passive roles cannot utilize. This

[18] This section covers the implementation from a rather high-level perspective and focuses on relevant aspects rather than detailed implementation decisions.

[19] The meta-model Figure 5.1 provided before already respected those terminological adjustments.

allows – as motivated in the original framework – the consistent use of agent notions, replacing objects even on the atomic implementation level. In consequence the two notions of micro-agents are now clearly split into one (higher-level) notion which communicates via asynchronous message passing, and a lower level one which still uses the method-based invocation which had been used for both levels in the original concept.

To utilize the feature set of the agent concept (and thus stronger agent-centrism instead of role-centrism) roles access agent capabilities using a reference to their agent (`getAgent()`) which is (mostly) hidden by the abstract role implementation (e.g. the default social role implementation `DefaultSocialRole` (refer to Figure 5.4 for an overview of the Role/Agent hierarchy)). Sending of messages (`send()`/`sendRandomcast()`/ ...) along with event subscription (`subscribe()`) and lifecycle management (`die()`) but also simple console output (augmented with a reference to the producing agent) (`print()`/`printError()`) from a role wrap this agent reference and ensure that according functionality can only be used if the role is actually played (i.e. initialized) by some agent.

Agents in turnaround hold references to all (played) roles including their applicable intents in order to allow dispatch of incoming messages[20] depending on the roles. Although holding this meta-information might appear redundant with regards to the lower Routing and Management Layer (MRL), maintaining this information as an inverted index provides the fastest possible lookup for this purpose.

Along with this special roles (building on the SocialRole) have been introduced to provide advanced patterns for message handling. One example is the `MessageFilterRole` which allows the definition of patterns for received messages upon which it executes defined actions. Message filters both support actions on positive matching (messages matching the pattern) as well as negative matching (message which do not match the pattern). The default implementation (`DefaultMessageFilterRole`) only provides simple but efficient value comparisons of message fields for a matching. Advanced pattern matching can be realized by extending the abstract MessageFilter class with own pattern matching implementations (e.g. using

[20]Here it should be recalled that messages are received by agents (see Figure 5.3), not roles directly (as with the legacy framework).

regular expressions). Upon initialization (via `addMessageFilter()`) a new separate agent is started and allocated as a sub-agent to the agent adding the message filter. By attaching multiple message filters agent functionality can be effectively (partially or fully) decomposed into sub-agents.[21] This way the realization of decomposition (in the understanding of AOSE) only requires minor implementation effort. This decomposition principle allows the cascading of functionality into sub-agents to an arbitrary depth. As a result decomposition with micro-agents is similarly simple as the semi-automated handling of groups (and thus the organisation aspect of AOSE) as well as abstraction (which is achieved by suppressing sub-agents from a developers view). Apart from this powerful feature message filters are the mean to achieve the compliance with the system-theoretical principle not to directly interact with other agents beyond the same subsystem. Their communication is always mediated by their 'owning agent'. Given this context, the use of message filters significantly enhances the potential to exploit the organisational aspects of the micro-agent model, easing its use while providing a decomposition mechanism which is consistent with system-theoretical principles.

Important to mention in the context of the message filters is the general message data structure, the MicroMessage. The MicroMessage class essentially wraps a HashMap with predefined accessors (e.g. `setRecipient()`, `setPerformative()`, ...) to allow a uniform data structure – only enforcing its keys to be of type String – which is firstly extensible with arbitrary fields and thus flexible for any kind of application but also allows to carry any kind of Java object as value. This is especially of relevance when sending intent and event objects but also allows to wrap binary data and encapsulation of messages themselves (as internally done in message filters). For the purposes of serialization (especially in the context of network communication) MicroMessages can be reliably casted to the HashMap core data structure without losing information (and technically even without necessity that the MicroMessage type is known by the deserializing party!).

Figure 5.4 shows a diagram of the core classes of the Agent Logic Layer of the μ^2 platform. A more extensive selection of class diagrams including elements of

[21] Example code for the use of Message filters in μ^2 is shown in Appendix A.2.2.

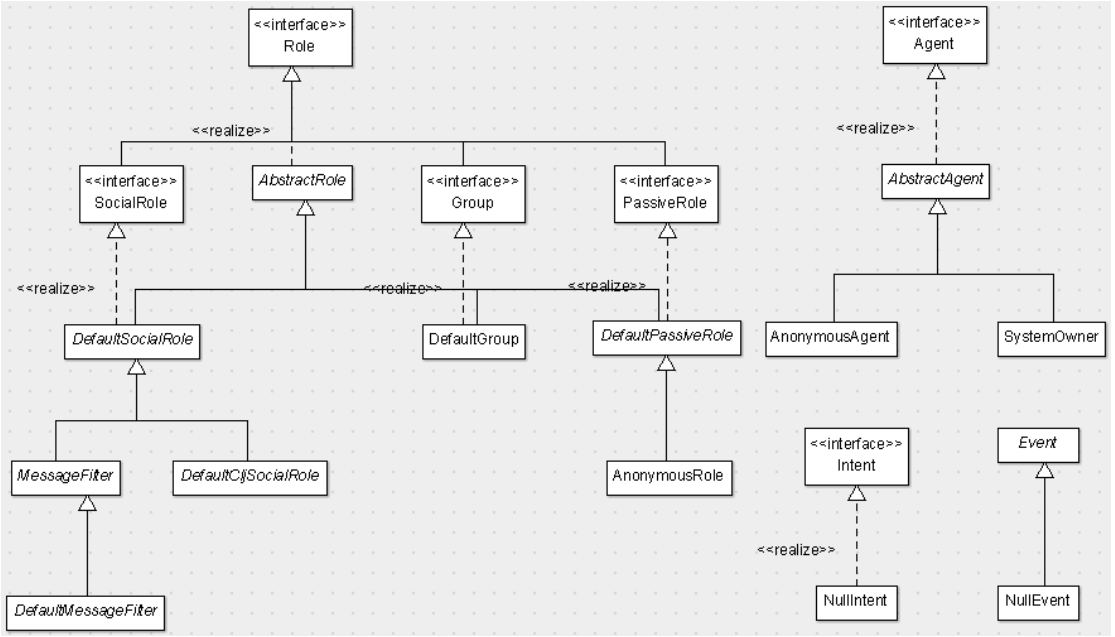


FIGURE 5.4: Class diagram of Agent Logic Layer of μ^2

both the ALL and MRL, the two top layers, can be found in Appendix A.3.[22]

Implementation of the Message Routing & Platform Management Layer

Implementation decisions done on the intermediate MRL involve the handling of intent and role directory for dynamic binding of targets. Similar to the handling of played roles on agent level, information is held in various index data structures using hash functions to allow constant lookup time[23]. The pseudo-code of the algorithm for message routing by intents and events (Dynamic Binding) is shown in Appendix A.1.

The detailed decomposition of the different addressing patterns (into either unicasts or broadcasts) is not discussed at this point but can be retraced from the framework code[24].

Additionally to the actual routing (and briefly shown in the mentioned algorithm)

[22]The extremely extensive MTL class structure is hardly of any added value without additional context (as of the integration of external libaries). The interested reader is referred to the actual source code respectively the platform website (see information in Appendix F.2).

[23]Here the problems of (time-consuming) rehashing and potential hash collisions are ignored.

[24]Information on the retrieval of the developed code is provided in Appendix F.2. Particularly the AbstractAgent and MTConnector class are concerned with the pattern decomposition aspect.

this layer allows the integration of a user-defined (or the provided default) message validator[25] which can perform message validation prior to the actual sending. Especially for less performance-sensitive applications, production environments as well as debugging during application development this can serve as a powerful feature as it potentially allows the integration of message logging.

A final relevant consideration at this stage is the implementation of the Randomcast which relies on a pseudo-random number generator to select agents for (potentially distributed) message dispatch. The default Java pseudo-random number generator is a Linear Congruential Generator (LCG) [LCG] and thus considerably simple and fast but has an unequal number distribution[26]. To ensure a better randomness which is a prerequisite for simulations an implementation (written by S. Luke [Luk]) of the Mersenne Twister [MN98] is integrated in the middle layer of the platform. Developers/users can supply the number generator with defined seeds to ease the replication of number sequences for debugging and verification purposes. The pseudo-random number generator can be used for arbitrary implementation purposes (e.g. role implementation, platform extension).

Implementation of the Message Transport & Platform Runtime Layer

The lowest layer demands for numerous implementation decisions rooted in the performance-sensitivity of this core layer of the micro-agent framework. Primary decision taken on this layer is the implementation of the message passing frameworks backing the internal agent communication. Basis for this decision is the review of selected frameworks as documented in section 4.3. Key criteria for the selection of message passing frameworks are firstly *usability/compatibility* from a software engineering perspective, secondly *performance*. A third aspect is the *fairness*. However, given the use of a dedicated (fair) scheduler (which is yet to be introduced) to execute the agent behaviour the latter is considered of limited concern.

[25]The default message validator simply checks for the existence of sender field (which is automatically set by the agent implementation) as well as the definition of either recipient, intent or event.

[26]The equality of distribution (randomness) depends on a bit's position. Especially lower bits have a poorer randomness (as nicely visualized by Neill Coffey [Cof]).

Based on those three objectives Jetlang was chosen as the message passing framework of choice as it does not enforce post-compilation code manipulation steps (such as Kilim which apart from this was nearly equally fast and achieved by far better fairness results). Introducing this 'weaving step' would not only put additional workload upon the developer (in the shape of an additional debugging step) but also prohibit real-time class compilation as necessary in the context of Clojure which relies on Just-in-Time compilation for the interpretation of its scripts respectively command line input. The downside of Jetlang, apart from the fairness aspect, is its approach of allocating one Java thread per agent which puts an additional memory burden on the system in the case of a large number of agents (such as in Massive MAS running several hundred agents). In order to avoid this another framework, MicroFiber, is developed which reuses elements from the Korus framework (and as such has a comparable performance). It provides a centralized scheduler which ensures by far better fairness[27] and delegates message processing to a specified number of worker threads[28]. The use of this framework is indicated if the platform exceeds a large number of agents and should be decided based on memory consumption of the JVM which is a matter of the analysis of a particular implementation (e.g. by profiling). As the MicroFiber framework delivers about half the performance of Jetlang, the latter should still be considered for the default use.

The consideration of the efficiency principle enforces a rather tight integration of the selected message passing frameworks without employing conventional abstraction approaches (e.g. Java interfaces) to ease exchange of the framework. As such both frameworks are tightly knit into this platform layer (as indicated in Figure 5.6) which is a compromise in favour of platform performance.

In the context of the network transport in-platform performance is of lower concern. The transport via the network itself enforces considerably more processing including (de)serialization. In this context core concern is rather compatibility than performance. As such the integration is mediated by an interface abstraction

[27]For details please refer to the benchmark results described in subsection 4.3.

[28]Tests with the framework parameters show best performance when the number of worker threads equals the number of CPU cores. As such the framework initializes this number if no other value is specified by the developer/administrator.

which allows easy introduction of alternative network transport mechanisms.
Similar to the internal communication network transport takes place in an asynchronous manner, making the use of Java NIO frameworks considerable[29]. In this context the framework of choice is the open-source solution Netty [Leea][30], partially based on the high performance promises indicated by various benchmarks (see [Leeb][31] and [Hea]) and its reliability as infrastructure of the well-known JBoss application server respectively middleware [JBo]. Specifically its rich collection of codecs for various purposes (e.g. SSL encryption, compression) and the support for Java serialization to ensure fully location-transparent use of the dynamic binding mechanism (via intents or events) are of interest in this context.
Beyond interoperability via the binary Java serialization – provided by the framework – additionally an XML-based serialization (backed by the XStream library [XSt]) has been developed to optionally provide human-readable protocols. This is considered for dedicated use when compatibility is of greater concern than performance (e.g. different Java versions, mobile devices, debugging).
Along with the actual communication functionality the platform runtime layer also manages network discovery which allows automatic discovery and connection establishment with remote platforms. This functionality is available in a dualistic fashion – similar to internal message passing and network serialization: Discovery is done using UDP Multicast functionality. UDP packets are sent to the according Multicast groups and contain necessary node information to allow connection initialization. The alternative approach is the use of Broadcast packets. Advantage of those is the stronger support by hardware. In fact the implementation of Multicast mechanisms is often both blocked on the according network or not supported by the according networking hardware. In contrast to Multicast the support for Broadcast is more mature.[32] The downside is the rather aggressive approach as

[29]Java NIO (or: New I/O) supports asynchronous I/O (not only for networking purposes but any kind of I/O (e.g. writing files)) and introduces the notion of channels in favour of socket-bound connections as part of Java specification 1.4. It is documented in JSR 51 [Rei].

[30]Its website provides rich documentation and receives active maintenance from the lead developer.

[31]This collection of benchmarks for different frameworks is developed by the Netty maintainer himself. However, all programme sources are provided to allow manipulation as well as replication.

[32]One example is the software Oracle VirtualBox [Vir] used for network tests. In fact only the Broadcast implementation worked sucessfully.

packets are eventually delivered to all network hosts[33] – whether or not nodes are interested in the according packets. A second issue is the binding to a local port which does not allow the use of Broadcast-based discovery for two platforms on the same host – a feature which works successfully using the Multicast implementation. Those considerations show, similarly to the trade-offs for internal message passing and network serialization, that a dualist approach is reasonable to provide a flexible platform which can be configured according to environment (other nodes, network configuration) and runtime application (number of agents).

The same approach has been taken for the aspect of configuration. Configuration of the platform can both be done via a XML-based configuration file as well as in-code if necessary. The latter allows a quick adjustment of parameters when useful. The XML-based variant is suggested for general use as of the separation of platform parameterization and implementation. Additionally the XML schema is compatible with the XML-based configuration files of the higher-level OPAL which allows a unified configuration mechanism. In-code specification overrides the values specified in the configuration file. A key risk in providing in-code configuration is the change of configuration at arbitrary places. This is prevented with μ^2; configuration changes are not applied once the platform is started. The configuration options are extensive (and among further options include internal message passing framework, network serialization, network discovery mode) and are driven by the motivation to increase application-specific performance by configuration rather than modification of application code (*tuning-by-configuration*).

Last but certainly not least aspect of the framework is its pervasive use of lazy initialization to allow the demand-oriented provision of capabilities as suggested for a weak wide agent notion (in the platform context (see subsection 2.1.3)). In order to keep the memory footprint of the platform (and as such agents) as small as possible, necessary infrastructure is only started once requested. In the simplest case neither internal message passing, network (transport and discovery) or Clojure are started. This would reflect the case of use if the platform does not make use of its communication capabilities (which is the case when using only

[33]Technically this is the same with Multicast. However, Multicast packets are filtered on lower level while Broadcast packets are passed upwards on the receiving host's network stack.

PassiveRole instances). The sequence diagram in Figure 5.5 visualizes this role-type-dependent lazy initialization.

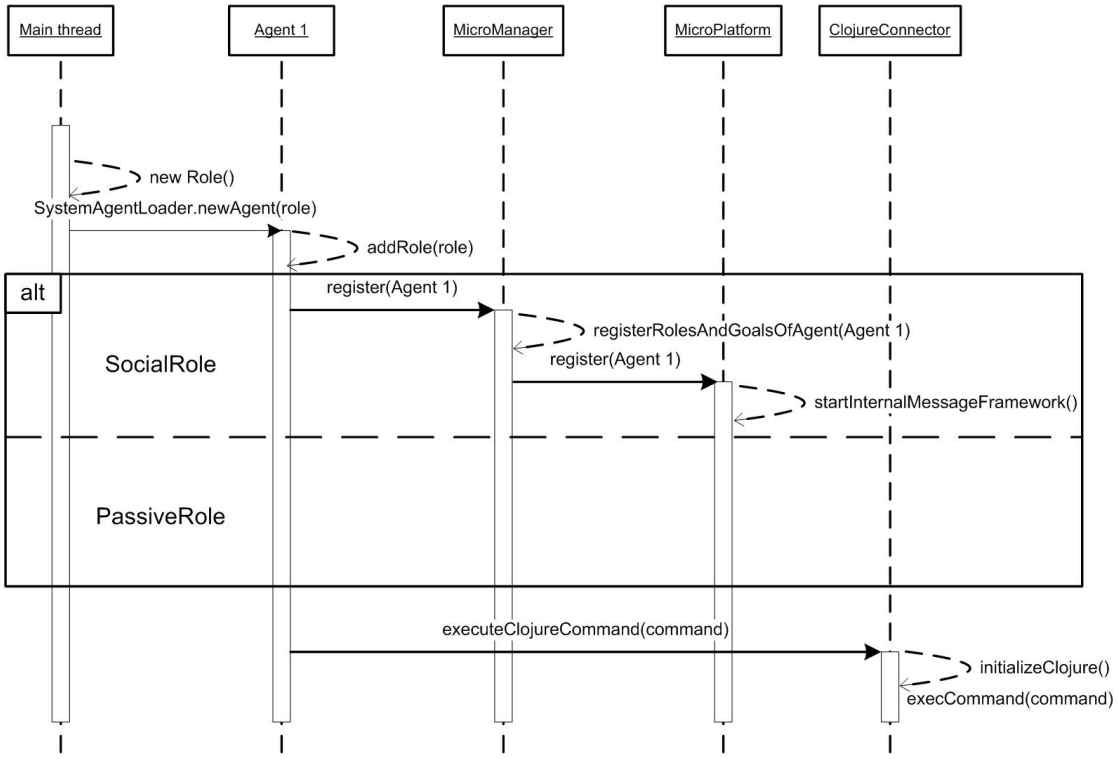


FIGURE 5.5: Lazy initialization of platform depending on role type

Performance Aspects and Summary

To test the satisfaction of the efficiency principle of micro-agents a simplistic agent benchmark is used to provide a comparison measure for the interaction performance of micro-agents in contrast to other multi-agent platforms. The scenario has been used in earlier works (see [NPC01]) and although the scenario is simple it still serves as a performance indicator and clarifies the strength of micro-agents. Its details are described in Appendix C along with the results. Some key figures are presented in Table 5.2 to allow better discussion with regards to requirements fulfillment.

The current performance of the reimplemented micro-agent platform is significantly lower than the original one (in fact takes about 2.5 times longer). Paying this penalty developers can communicate asynchronously and are freed from any thread handling. Considering that MadKit, a framework considered of the same performance cluster as μ^2 (see Appendix C.2), still has a penalty factor of another

Platform	Runtime (in ms)	Relative difference to μ^2	Runtime over network (in ms)
KEA micro-agents	170	0.41	–
μ^2	410 (Jetlang), 800 (MicroFiber)	1 (Jetlang)	12000
MadKit	1025	2.5	(not tested)
JADE	8303	20.25	(not tested)
OPAL	10400	25.36	130770

TABLE 5.2: Benchmark results for Agent Platforms relative to μ^2 for 10000 rounds

2.5 this performance is considered reasonable to still fulfill the efficiency principle of micro-agents. The linear performance with increasing benchmark rounds (see Figure C.3 in Appendix C.2) rectifies this view.

On the network side the use of the well-performing underlying message transport pays off in shape of a significant performance advantage in contrast to OPAL. OPAL provides stronger levels of message transport abstractions and accepts the performance penalty involved when complying with the FIPA communication standards. The micro-agent network capabilities are thus useful if performance is of primary concern – networking in OPAL takes nearly 11 times as long (for the benchmark case). The fact that network communication via the micro-agent framework is nearly equally fast as internal communication in OPAL makes a clear case to suggest a careful selection of an appropriate framework for the according application context.

Some of the implementation decisions described above are summarized in Figure 5.6 which provides a implementation-oriented view on the design described in subsection 5.2.1. Examples showing the implementation of agents with the reimplemented agent platform (i.e. the equivalent to the simple interaction example shown in subsection 5.1.1 and the use of message filters) are provided in Appendix A.2.

Summing up, the reimplementation of the micro-agent framework elaborates it towards a fully-fledged platform, which leaves the actual agent meta-model on the highest level and breaks the functionality down towards the lowest level (decomposition of communication patterns into unicast/broadcast respectively local/remote

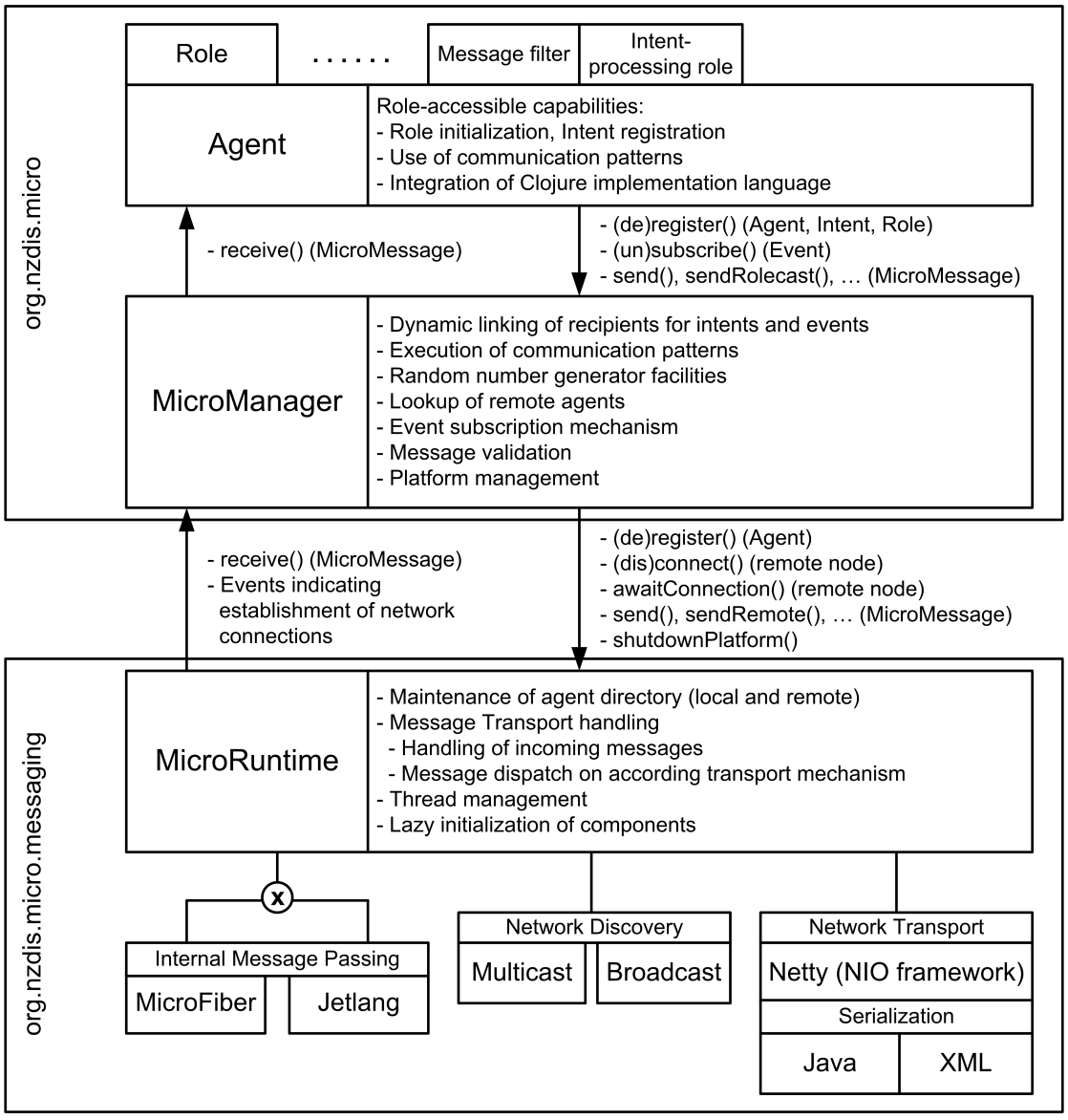


FIGURE 5.6: Implementation-oriented schema of platform layers in μ^2

on intermediate layer) into a pure performance-orientation on the lowest level (actual communication, network integration). In contrast to the legacy framework it also integrates pervasive network support which makes it irrelevant for the developer whether his applications are distributed or not. Additionally the platform learns a lesson from the area of simulation, and takes randomization mechanisms into account at various places. The reimplemented platform supports the application developer with powerful configuration mechanisms which leaves room for the mere focus on modelling in an agent-oriented manner without infrastructural concerns and a fully automated dynamic binding of agents. Apart from this the

platform has an impressive performance in the field of MAS for AOSE and yet seeks for an equivalent in this respect.

The various new properties of the platform inhibit one of its characteristics, the inherent duality of its components for the sake of flexible use: It provides two alternative message passing frameworks, two general types of network serialization, two discovery mechanisms and last but not least two options for configuration. This along with the fact that it is the successor of the legacy micro-agent framework constitutes its name μ^2.

5.3 Additional Platform Extensions

5.3.1 Clojure as Agent/Environment Implementation Language

For the integration of Clojure as role implementation language[34] several aspects are considered. As the encapsulation of the agent internals is a primary principle (and can be well represented using objects in the case of Java), Clojure enforces the use of different mechanisms. In fact using Clojure the developer (and even the actual application user) can access all object instances loaded into the same JVM at runtime (in the general case all properties and methods with public access modifier, in the extreme case all internals via the powerful Java reflection). Even when instantiating/accessing Clojure from different objects they share the JVM memory; Clojure role implementations could directly access internal state of other agents. To avoid this (and assuming that all agents are benevolent by not mutually accessing their internals) Clojure namespaces are used to represent separate agent respectively role instances (i.e. agents operate in their individual namespaces).

Considering the intimate relation between agent instance and Clojure namespace, the integration of Clojure as implementation languages is done on Agent Logic Level, allowing the use of all agent capabilities (certainly including sending messages) by role implementations written in Clojure. As of the unified message container this does not restrict Clojure roles from interacting with roles written in

[34]Please recall that the application developer implements roles rather than agents (although the system should now equally allow the latter (see subsection 5.2.1)).

Java which makes it a valuable alternative for role implementations, even on the same micro-agent. To be able to receive messages sent to its agent, roles implemented in Clojure merely need to provide a `receive-msg`[35] function.

More implementation-related details in Clojure can be found in the context of the platform itself (particularly the CulturalIndividual.clj script as indicated in Appendix F.2). Information on more general aspects like the practical use of Java interoperation can be found under [Clo].

Conceptually Clojure does not only show potential in the context of agent or role implementations but also as environment implementation. Its particular strength in the context of concurrency is its STM which allows concurrent manipulation of shared memory and provides a consistent view on it at any time. This is the case for environments in general anyway but seems particularly interesting in the case of social simulations with a generally stronger emphasis of an environment (see subsection 3.2.3). Figure 5.7 visualizes this design which can serve as a pattern for the implementation of environment-centric MAS. It suggests the use of message passing frameworks for direct communication and a STM-backed environment (in this case with Clojure) for indirect communication.

In a practical application this would include the allocation of a namespace for the environment and adding a reference to the environment namespace to every

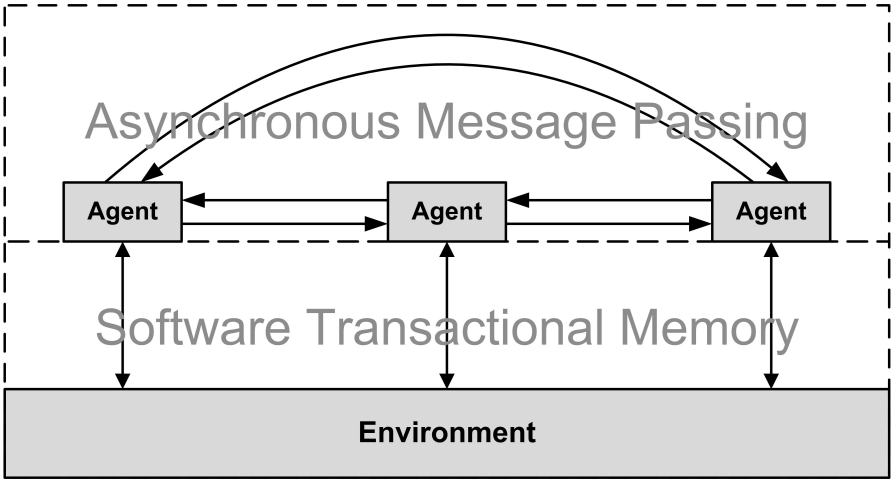


FIGURE 5.7: Reference design for social simulations in μ^2

individual agent/role namespace – thus satisfying the separation of concerns while

[35]The function in fact needs to have one parameter which will eventually be the received MicroMessage.

accessing a common environment. An example/test application built on this design is the 'TalkingAnts' simulation which described in Appendix B and had been iteratively extended to serve as a benchmark in the context of the scheduling component.

In conclusion, Clojure in μ^2 can be used for any component of a MAS or simulation application and also allows the complementary use along with Java as 'native' implementation language.

5.3.2 Fair Scheduler

A necessary element to control the limited fairness provided by message passing frameworks is the scheduler which is provided for optional use in the case of autonomous agents. Its primary role is to provide round-level fairness in a cooperative manner[36] between all scheduled tasks. For the design originally two approaches had been considered. A consideration to use the in-built Java functionality in the shape of thread pools was abandoned as those merely serve as a wrapper to control regular and preemptively scheduled Java threads – which is unfair on a round-level basis (considering the potentially unequal execution time of agents). Another concern is the limited scalability. Java thread pools allow the submission of so-called 'Runnables' which (for the given context) can be seen as tasks. However, for each submitted Runnable a new thread is started[37] which leads to higher memory consumption and a poor scalability behaviour for a large number of tasks (one thread per task) as of the significant overhead involved in context switching between all threads by the JVM scheduler.

The first alternative to default Java scheduling behaviour involves the introduction of the notion of cooperative threads on JVM level to allow a general fair scheduling. For this purpose the use of the *FairThreads* framework [Bou], born as part of the research effort around Reactive Programming by the French *Institut National de Recherche en Informatique et en Automatique* (INRIA), had been considered. However, pretests with this framework showed an effectively sequential execution

[36] Each agent can occupy a thread as long as necessary to complete its round of activity but eventually needs to give up control of the thread (*cooperate*) in order to allow the system to schedule the next round.

[37] Runnables submitted to a thread pool which is initialized with a fixed size (and whose threads are busy) are simply not executed at all.

of threads – even showing performance penalties in contrast to single-threaded round robin scheduling as of the context-switching between the threads.

As a consequence an own application-level task scheduler is suggested (FairTaskScheduler) which can be optionally used (it might not be of use in purely reactive MAS applications or conventional MAS applications in general) and is less intrusive in contrast to threads as it introduces a light-weight notion of tasks (rather than threads). It ensures round-level fairness and in order to achieve this has multi-threaded round robin semantics. A user-defined number of worker threads[38] process task queues in a concurrent fashion until all tasks have completed the current round.

In order to evaluate performance and fairness the 'TalkingAnts' application – introduced in the context of the Clojure integration – has been backed with the different schedulers. The benchmark is described in Appendix B.2 along with the benchmark results. The results clearly indicate the lack of performance of the FairThreads framework, achieving about half number of rounds as compared to the FairTaskScheduler. The FairTaskScheduler achieves about two thirds of the performance provided by a pure Java Thread Pool implementation. However, the thread pool shows a steadily increasing deviation from an equally distributed execution on round level, showing the limited fairness provided by Java. This clarifies why the use of a dedicated scheduler is unavoidable when considering the execution of both valid and replicable results.

Further features of the FairTaskScheduler – apart from the scheduling itself – include both a continuous mode which schedules rounds continuously and a stepping mode which allows the execution of a user-defined number of rounds, along with its specialization to run for a specified time period (e.g. 30 minutes). This latter mode is particularly targeted towards experiments.

5.4 Micro-agents on Android (MOA)

An area which has only been implicitly touched is the provision of an Android-based version of the micro-agent platform.

The approach taken here is divided in two steps, firstly the actual 'translation' of

[38]The default number is the number of CPU cores of the according machine.

the Java version of the platform to Android, and secondly, the integration with Android's concurrency model to not only allow user control of the micro-agent framework but also the transparent interactive access of Android functionality by micro-agents – in fact similar to the interactive access of Java functionality from Clojure. As the latter part is the conceptually more valuable one the focus is consequently put on this.

5.4.1 Porting μ^2 to Android

The actual 'translation' of the Java version to the Android platform is done iteratively. This decision is taken after initial pretests revealed that many critical Java core libraries (especially concurrent collections, necessary for the support of thread-safe asynchronous message passing) were available. In consequence the development primarily concentrated on the desktop version until a stable feature set had emerged. From this point onwards the desktop version has been ported to the Android environment, adjusting all differing libraries[39] either by removal of functionality or replacement with available equivalents provided by Android. In consequence each further development iteration of the desktop version included a refinement of the ported version where necessary. This way the code base is identical where possible.

Major areas relevant for μ^2 where library support by Android differs or is lacking are discussed below:

- *Lack of Just-in-Time (JIT) compilation*
- *Incompatibility of native Java serialization*
- *Differences in network libraries*

The lacking JIT compilation proves to be the most significant drawback of Android for the micro-agent framework. As a consequence the integration of Clojure support for Android had to be delayed as it heavily relies on JIT compilation to allow its interactive programming and REPL-based evaluation. An alternative is the compilation of Clojure code ahead of time (which would render it in regular Java byte-code) and running it on Android, as already realized by an open source

[39] Please recall that although Android uses Java as its application implementation language and provides a wide range of well-established Java libraries not all libraries/interfaces are supported or compatible.

project (see [vtV]). However, taking this approach Clojure would merely be a development vehicle but not allow the processing of Clojure statements at runtime (e.g. passed in messages) which is the striking feature for its use in the context of open systems and interactive development. Although developmental steps to provide a JIT compiler are taken (see [Wim], [Goo10]), those are yet in an immature state harming the robustness of the overall system. As such the integration of Clojure is delayed until an equally robust mechanism is available for Android.
The serialization between Java and Android proves not to be fully compatible and harms the aspect of interoperability between desktop version and mobile version of the platform. As the network communication framework Netty allows the integration of a custom Java serialization (which only partially relies on the underlying Java mechanisms), it is introduced both in the mobile version and – as part of the developmental feedback cycle – into the desktop version. Additional to this an alternative XML serialization is provided (as stated in subsection 5.2.2) to allow the provision of open systems as well as a fall-back in case of incompatibilities in the binary serialization format (for the sake of debugging). Using the mechanisms described here, the communication between desktop and mobile version can be seemlessly established and Java objects exchanged.
The third consideration is the difference in the network socket libaries between Java and Android. While persistent connections for means of transport are largely delegated to the Netty framework, the network discovery in μ^2 is purely developed via socket programming. As Android's semantics to gain access to sockets partially differ[40], this part of the application has been entirely rewritten. In consequence the network discovery in MOA is functionally compatible (but not code-compatible) with the desktop version of the platform and thus allows mutual platform discovery, a feature particularly appealing on mobile devices where any need to enter ip addresses or the like should be avoided.
Summing up, the development of the Android port of the actual platform is predominantly focused on the adaption of the implementation, relying on the design produced in the context of the desktop version. This way nearly full portability

[40] An example is the WifiManager to gain direct access to the (wireless) network.

of micro-agent code on Android and Java – apart from the Clojure support – is provided, including the network interaction between both platforms.

5.4.2 Interfacing Micro-agents with Android

The more appealing part of the development involves the integration of the micro-agent platform with the Android communication mechanisms which is introduced from a considerably high-level perspective.

Bearing in mind the loose coupling of application components (such as Services, Activities, BroadcastReceiver and ContentProvider as introduced in subsection 4.2.2) by means of intents which allow the operation in open application environments (i.e. unknown environment with regards to installed applications on the device instance), the interaction between both micro-agent platform and Android itself provides interesting opportunities. Micro-agents could thus directly interact with Android components such as activities or services in an asynchronous manner. Examples for the capabilities could be the use of phone book data or available location information (i.e. GPS data) and associate those with received SMS to provide context-related automatic replies or to combine those in order to develop decentralized agent-based location-based services or simply enable agents with the ability to send and receive SMS (e.g. as fall-back mechanism if alternative network connections are lost). The realm of applications opened by this perspective is wide. The openness of the Android environment makes the embedding of agents in general and micro-agents in specific attractive as their particular strength is the handling of the inherent non-determinism within environments. This effectively realizes the open system principle on mobile platforms.

Similarities between μ^2 and Android

As an initial approach related concepts of both systems are identified to design a potential mapping of components.

A core similarity is the use of *intents* for communication. In both cases intents describe an abstract task request but differ in various implementation aspects. While intents in μ^2 are statically typed at development time (e.g. PrintIntent as intent specialization), Android's intents are dynamically typed, and thus defined

at runtime. In contrast to the micro-agent platform intents in Android additionally serve as message containers. In μ^2 intents are wrapped into MicroMessages which represent the actual message containers. Along with this Android's notion of *intent filters* (which allow the resolution of implicit intents) maps considerably well on the notion of *applicable intents* associated to micro-agent roles which are used to resolve requests to according agents respectively roles.

Apart from this the concept of micro-agents is loosely related to *Services*[41] as – equally to micro-agents – Android services are running in the background and often have long-running tasks (potentially throughout the system uptime) and typically operate asynchronously. However, they only fulfill those runtime qualities. Micro-agents inhibit goal-directedness and management of ongoing conversations which is beyond the scope of Android services.

Components with higher visibility to potential users are the actual Activities which are often augmented with a graphical user interface and are rather short-running. From this point of view they loosely relate to agent operations which eventually have impact on the application user. This tie is considerably loose as operations as such are a largely implicit concept in μ^2.

Last similarity is the event subscription mechanism in μ^2 with Android's BroadcastReceiver mechanism. Similarly to the (micro-agent) intent, events are statically typed and passed via MicroMessages while Android simply broadcasts its intents.

Considering those similarities Android in itself provides infrastructure which is the basis for a multi-agent system as such and certainly an appealing basis for integration and interoperation (as described earlier). But still Android in itself does not qualify as a multi-agent system.

Table 5.3 sums up the observations described in the previous paragraphs.

Design of an interface mechanism

In order to interface between those related technologies the according internal mechanisms need to be mapped.

In the micro-agent package intents are encapsulated within a unified message structure – and are not message structure themselves. Further contrast is the static

[41] At this point it should be reemphasized that the term refers to the Android concept, not to service-oriented technologies or common understanding.

μ^2	**Android**
Intent – request specification mechanism (but not message container)	*Intent* – request specification mechanism and message container
Applicable Intent – Specification of applicable intents executable by micro-agent role	*Intent filter* – Functionality specification of application component for request resolution
Micro-agent – composable persistent runtime entity	*Service* – Wrapper for long-running (asynchronously executed) background task
Operation – short-running (often visible) agent actions	*Activity* – short-running task (often with direct user interaction)
Event – message structure used by event subscription mechanism	*Broadcast* – System- or user-defined intents sent as broadcast for reception by registered broadcast receivers

TABLE 5.3: Related concepts of μ^2 and Android

typing in the case of the micro-agent platform. In consequence a special (micro-agent) intent resembling the method signature of Android's intent has been designed (and named `AndroidIntent`). This way micro-agents can directly specify the contents of an intent for the Android environment but are independent from the Android platform themselves (i.e. can reside on a different host). Only the specialized data structure needs to be accessible by micro-agents and is thus packaged with any version of μ^2. As the conversion between the interface-compatible (but not semantically compatible) intents cannot be performed on non-Android operating systems, this task is ultimately linked to the Android version of μ^2, *Micro-agents on Android* (MOA)[42]. Along with the conversion of the 'intents' the transition between both runtime environments needs to be established.

To realize this the micro-agent framework running on Android is managed by an Android service. As services are long-running and reside in the background (in contrast to activities) the use seems most appropriate to manage MOA. The Android service itself is intimately linked to a micro-agent (or more precise, an initialized role instance by means of static references) in the micro-agent framework which offers the conversion mechanism and uses the linked service to dispatch the

[42]The name is not only an acronym but refers to the now extinct giant flightless native New Zealand bird – in fact less mobile than Android nowadays. The largest bone collection is held in the Otago Museum in Dunedin, making the reference to this species only but consequent.

converted intent into the 'Android world'. To manage the 'conversation' between the two realms Android intents are supplied with sufficient metadata to associate a potential response (from Android) with the original sender (micro-agent). During conversions several metadata fields are thus appended to the extensible Android intents. As the micro-agent role linked to the Android service registers AndroidIntents (i.e. micro-agent data structure resembling actual Android intent) as its applicable intents (i.e. as requests which can be fulfilled by this role), the dynamic binding mechanism on the micro-agent platform automatically ensures that all messages containing an `AndroidIntent` will be sent to this according role.

In consequence micro-agents are able to directly interact with Android components and applications via intents. Downside of this approach is the need to have knowledge about the basic communication mechanisms respectively the Android concept of intents in order to formulate those. An alternative would be to essentially wrap desired Android functionality as artifacts (such as SMS message or phone book) and invoke those using the generic communication mechanisms of the micro-agent framework, such as done in JaCa-Android [JaC], or fully build an agent framework using the Android application components as infrastructure, such as done by Agüero et al. [ARCJ09]. However, for the first case the developer has less flexibility as any potentially interesting resource would need to be encapsulated ex ante which would rule out the perception of Android as an open system (from the perspective of the agent platform). Along with this it shall be recalled that micro-agents in fact focus on efficiency and establish connections to low level resources. The direct conversion into intents represents the most efficient – and from the point of micro-agents – the conceptually suitable way to interact with Android. Building an multi-agent system natively on Android infrastructure, as suggested in the second alternative, would additionally harm the seemless interoperation between desktop and mobile version of the platform and would harm performance – as to be shown later.

Not yet described is the opposing perspective – the interaction of Android components with micro-agents. One simple aspect is the subscription to Android system events. Events sent by the Android system can thus be processed by micro-agents

for arbitrary purposes[43]. In order to achieve this the Android service wrapping the micro-agent platform needs to subscribe according events (using a BroadcastReceiver component) and delegate those to the micro-agent platform which are converted and raised as events micro-agents can subscribe to.

Another approach to allow active invocation of micro-agent facilities from Android is the use of intent filters. To enable this (similar to the registration of *applicable intents* on MOA) the service wrapping the micro-agent platform needs to register according intent filters and as such could replace core system services or applications of Android itself (such as the phone application) and model it with micro-agents.

An example scenario for the use of MOA, describing its potential and visualizing the internal architecture, is provided in Appendix D.1.

Potential application areas

Considering the areas of applicability, the connection between those two technologies (along with the principle of efficient and 'cheap'[44] micro-agents) mutually opens up several application areas respectively provides advantages:

- *Robotics* – Android as such is attractive for the area of robotics as of its communication capabilities, open source availability and considerably moderate hardware demands. Combining it with micro-agents allows the development of robot application based on micro-agents (including higher-level planning/reasoning abilities) with built-in communication and sensing capabilities (e.g. robot communication via SMS, use of the provided camera(s), GPS-based location information). Alternative approaches would include the possibility to externalize the agent logic into Android devices connected to the mechanical robot (respectively its actuators and sensors) via Bluetooth (i.e. 'externalize the brain' of the robot). The logic could then be physically decoupled from the robot hardware, allowing the use of inexpensive

[43] A simple scenario highlights the potential: Received events such as an incoming call could result in automatic redirection of the call to the mailbox if (the phone) moving beyond a certain speed (assumed to be driving in car) and agents could automatically respond per SMS ("Dear XY (resolved via contacts on phone), cannot answer your call as I am driving.") without any user interaction.

[44] Micro-agents are cheap from a resource point of view (i.e. low memory consumption).

robot technology rather than specialized microprocessors to run necessary programming environments (e.g. JVM).

- *Intelligent Agent Applications* – Applications involving intelligent agents could be built based on the micro-agent platform and directly access Android capabilities. This does not only allow more complex agents to run on mobile devices (as provided with other platforms) but also use available resources of semantic value (e.g. reasoning could consider phone book entries or available media data). For other solutions this kind of functionality would need to be specifically developed; the suggested design delivers these capabilities out of the box.
- *Agent-Based Middleware* – Android can make use of MOA in an inverted relation and use its strong network capabilities (including communication with desktop machines) as a middleware for general development of software for both desktop and mobile devices. This goes beyond the vision of AOSE but shows an effective (and probably efficient) use of agent-based technology to stimulate the convergence between desktop and mobile world.
- *Performance* – A less spectacular but surprising consideration is performance. Although both the micro-agent platform and Android are built on similar principles, MOA achieved a significantly higher performance as documented in Appendix D.2. Partial reason for this is the more light-weight agent entities and strongly efficiency-oriented messaging framework. As such application logic could be delegated to micro-agents as far as useful or possible and rely on Android intents only to actually access Android resources or to offer GUI interaction.

5.5 Summary

This entire chapter presents a comprehensive overview over the design and reimplementation of the actual micro-agent framework by first pointing out the limitations of the original implementation and defining requirements for a reimplementation. As of the number of requirements and the significant changes to the technological foundations and feature set a reimplementation is inevitable. Consequently the design for the successor, μ^2, is described. It pragmatically loosens the core

relations of the original model and introduces a layer model below the actual agent layer which results in a transition from a framework to a platform. Characteristics of it include the comprehensive management facilities for both platform and agent lifecycles along with an integrated and comprehensive network support. On the other hand μ^2 strengthens the compliance of the multi-level nature with system-theoretical aspects in contrast to its predecessor and eases the modelling of decomposition in agent-based applications. Apart from the powerful decomposition (and abstraction) mechanisms, the platform enhances the emphasis for flexible communication. Agents, respectively roles, can interact either by dynamic linking via intents or numerous addressing patterns to facilitate specific modelling needs.

Along with this Clojure has been integrated and does not only provide a further interoperable implementation language but also an alternative concurrency handling mechanism which provides a basis for the implementation of agent environments. Furthermore a fair multi-threaded round-robin scheduler has been added to support necessary fairness. Various aspects such as the Randomcast addressing, the explicit consideration of an environment as well as the fair scheduler, target the use of μ^2 in social simulations.

As a further outcome the port of μ^2 to the Android platform shows its usability on mobile devices but even goes further as it enables the developer to directly access any phone capability and vice versa allow the use of MOA as a middleware for Android applications itself, offering both a performance advantage as well as network interaction with the Java version of μ^2. It offers further potential for the area of robotics and the use of 'intelligent' agent technology on mobile devices. Although this aspect will not be further explored at this point, it indicates a significant conceptual step towards the realization of open systems on mobile devices. Combined, those results serve as an argument for a better acknowledgement of agent-based technologies for general software engineering purposes.

Chapter 6

Simulation Scenario

In order to show the usability of the developed platform along with its particular strength in the area of interaction a social simulation scenario is modelled and implemented. The scenario of choice involves a strong degree of interaction between numerous agents and exploits some key features of the new platform (e.g. Clojure, asynchronous message passing, fair scheduling).

The idea to supply agents with more human-like attitude makes the incorporation of cultural aspects into group forming mechanisms attractive. This model is rooted in the understanding that many phenomena of interest in the social sciences can be reduced to a group shaping mechanism – independent from kind of groups or their persistence (e.g. finding trade partners, developing organisations). A popular example clarifying this principle is the Schelling model of segregation [Sch71] which describes the movement of agents around a grid until they reach a maximum degree of 'happiness' which ultimately ends up in clearly segregated clusters. In this context 'happiness' is correlated to the degree of homogeneity of an agent's immediate surrounding with respect to skin colour. Apart from the group forming principle the model described here has considerable differences as agents do not actively move in an environment but actively engage in negotiations – thus show advanced social behaviour in the sense of both MABS and AOSE (see discussion in subsection 2.1.1) – instead of communicating by direct introspection of neighbouring agents' minds.

6.1 Scenario Background

As a theoretical framework this scenario uses concept of *Cultural Dimensions*, elaborated by Geert Hofstede [Hof01]. Although approaches to categorize culture are documented (e.g. Todd [Tod83]), those either do not sufficiently abstract from ethnocentrism[1], are merely typologies (such as Todd) or lack empirical backing which limits their applicability. Typologies, in contrast to dimensions, try to align empirical observation to distinct idealized types which may often hardly match and do not catch cases of hybrids; dimensions have those idealized types in their extremes but allow a combinations to match observations in a more flexible manner (see Hofstede [Hof01], p.28f.)[2].

A key advantage of Hofstede's dimensions is the empirical backing for a considerably extensive set of countries. All observations were done in IBM subsidiaries throughout the world in two rounds between 1967-1969 and 1971-1973 as part of an attitude survey, a characteristic feature of the company culture.

The survey initiated in 1967 was the first internationally standardized questionnaire of the company (covering 180 items); both surveys covered IBM subsidiaries in 72 countries and 20 languages. As part of the results Hofstede elaborated four cultural dimensions, scored by means of index values:

Power Distance[3] indicates the degree to which subordinates accept the unequal distribution of power respectively fear of disagreement with their superiors. High values indicate high acceptance of the inequality.

Uncertainty Avoidance describes the degree to which cultures (in Hofstede's context factually countries) prepare against the unexpected – be it by means of technology (controlling unforseen natural impacts), law (controlling behaviour) or religion (controlling the uncontrollable) – or simply accept uncertainty. Indicators for high degrees of uncertainty avoidance can thus be strong rule-orientation in behavioural patterns.

Individualism opposes collectivism and indicates in how far a culture's thinking

[1] Hofstede describes this problem with regards to researchers lacking cultural awareness (such as the unreflected cross-cultural use of Maslow's hierarchy of needs [Mas43]) in [Hof01], p.17ff.

[2] Typologies have discrete character while dimensions emphasize a continuous understanding.

[3] This term is in fact suggested by Dutch sociologist Mauk Mulder but has been empirically backed by Hofstede (see Hofstede[Hof01], p.83).

reflects a collective thinking or an individual-centered self-understanding. Groups in collective societies are typically of larger size, while their counterpart is driven by the notion of volatile groupings and the nuclear family notion.

Masculinity describes the degree to which the attitude of members of both genders in a society are leaning towards principles either ascribed to masculinity or femininity.[4] Societies with stronger focus on masculinity are rather driven by competition and a work-centered lifestyle in contrast to an orientation towards quality of life and stronger focus on relationships (with a tendency to smaller companies).

Long-term Orientation is a dimension introduced at a later point in time and was adopted by Hofstede from a Chinese value study conducted by Bond [HB88]. The latter study seeked to complement the other dimensions – driven by a 'Western' understanding – by introducing the notion of 'confucian dynamism' respectively long-term vs. short-term orientation. Cultures with high long-term orientation favour perseverance over fast results, put less emphasis on the past but future life. With regards to Asian countries it is largely seen as a complement to uncertainty avoidance (see Hofstede [Hof01]).

Characteristics of the dimensions exceed the simple explanations given here by far. Details on data collection and validation as well as correlations between dimensions (e.g. power distance values and uncertainty avoidance correlate for Western countries) are described in detail by Hofstede [Hof01].

The cultural dimensions and their data base (employers of one multi-national company) have been target to criticism, along with Hofstede's considerably static understanding of culture as 'software of the mind' with a considerable change resistence over time (see McSweeney [McS02]). Apart from those aspects, the striking reason to apply its understanding are the extensive empirical observations which allow to model actual cultural stereotypes.

The focus of the model described here is not to achieve this in an accurate manner but rather to show one approach to operationalize the cultural dimensions and to show how culture affects group shaping mechanisms. In contrast to an earlier (but

[4]Here it should be mentioned that Hofstede himself implicitly applies gender roles or stereotypes to classify cultures.

considerably recent) application of agent-based modelling to cultural dimensions[5], the approach taken here targets a large number of agents and can be general basis for further models with more orientation towards specific problems (e.g. strategy selection based on cultural background).

6.2 High-level Model Description

In the following the conceptual model of the simulation scenario is described in more detail and key aspects of the framework by Gilbert and Troitzsch [GT05] (as introduced in subsection 3.2.2) are applied.

The real-life phenomenon to be modelled are the groups which emerge based on the autonomous interactions between individuals of arbitrary cultural background. Relevant aspects for observation include potential patterns of emerging groups with regards to their cultural composition which are explored for other (unexpected) emerging characteristics. Characteristics of both in- vs. out-group individuals shall be observed to gain better insight into individual agent decisions. Underlying core assumptions for the modelling process are:

- Every individual has a motivation to join groups (at least at some point in time).
- Individuals cannot directly perceive any cultural dimension characteristics (or cultural background) of other agents but seem to 'know' if somebody is of their culture.
- Group memberships are explicitly negotiated.
- Individuals of a given culture show deviations from their 'mean' cultural characteristics.
- Individuals only belong to one group at the same time.
- Groups do not merge.

Although the assumptions might be target to discussion they shall hold for the model described here. Additional assumptions (such as the understanding of the cultural dimensions themselves) will be introduced in subsection 6.3.

The model thus consists of so-called cultural individuals which hold characteristics

[5]Hofstede, Jonker and Verwaart [HJV09] suggest a model for intercultural trade relationships focusing on few BDI-like agents to perform trade transaction steps.

representing the three cultural dimensions *Individualism (IDV)*, *Power Distance (PDI)* as well as *Uncertainty Avoidance (UAI)*. As the cultural dimensions can be applied in isolation of each other (i.e. can be interpreted independently), a selective use is possible and also eases the interpretation (as of the fewer independent variables). Apart from this the operationalization of Masculinity (MAS) and Long-Term Orientation (LTO) is possible but less intuitive with regards to the problem observed in this model and would raise the complexity of the model significantly (both in terms of 'understanding' and processing).

One core aspect of this modelling approach is that the modelled cultural characteristics cannot be explicitly perceived by other agents.

Apart from those cultural characteristics individuals also inhibit *four personal characteristics* which describe externally perceivable characteristics. Those *could* (and are named as this for the sake of intuitive use) represent ethnicity, gender or language[6] but are in fact *do not carry any semantic association* and are merely used for determination of perceivable differences between individuals, and as such helpful for their discrimination. Additional to this a further externally perceivable individual characteristic *with a semantic association* is introduced, namely (social) *status*.

The individual behaviour is modelled as following and executed in the shape of rounds – ensuring an equal chance of each individual. Ungrouped individuals initially determine whether they are interested to join a group. If this is the case they request group information (group average values for status and personal characteristics as well as group size) from randomly selected agents. Upon provision of this information, agents deliberate which group is most appealing for them and request the membership. Depending on the desperation of agents they will consider more or less groups before applying. Groups equally deliberate whether or not to accept the applicant by evaluating the benefit based on the individual's perceivable characteristics and potential earlier applications or group leaves. Ungrouped individuals which receive a group information request can respond as a group of size

[6]Those examples in fact carry some 'cultural baggage' but are hardly sole criterion for a culture. Their use in this model is inconsistent (i.e. no dependencies between the randomly chosen personal characteristics are respected).

1. Ungrouped individuals which receive a request for group membership can then start an actual group (as group leader) if they accept the membership request. Individuals which have been rejected remember this rejection and need to await the next round to apply to a new set of groups. Individuals who have not been interested in joining a group (based on their attitude) become more 'desperate' to do so which – at a given point – will overrule their lack of interest and makes them participate in membership requests.

Individuals which are grouped behave differently depending on their role as members or group leaders.

Members are free to decide whether they want to leave the group (e.g. because of new members) or to remain. Group leaders have the option to decide whether they want to expel a certain member which does not fit into the group (e.g. because of individuals whose properties are outlier properties). In both cases individual and group remember their counterpart (i.e. group remembers leaving individual, expellee remembers group it was forced to leave) for future deliberations.

6.3 Operationalization of Cultural Dimensions

Up to this stage the simulation case has been presented on a high level; the according operationalization of the mentioned characteristics is not specified yet.

The original cultural dimensions are measured on a positive scale in integer steps originally ranging from 0 upwards[7]. As high respectively low values for all concerned cultural dimensions potentially show opposing behaviour in the given context, their value distribution is normalized around the value of 0 which indicates a neutral attitude with regards to a given dimension. The values assigned to individuals range from -5 to 5.

The understanding of the cultural dimensions in the given context thus needs to be translated to 'negative' for low values (in Hofstede's concept) and 'positive' for high values. Following this the operationalization of the dimensions will be explained bearing this understanding in mind.

For the power distance (PDI) a general dependence on the social status of individuals or groups is considered relevant. A positive PDI thus indicates a strong

[7]The highest value found describes the Long-Term Orientation in China (value 118).

acceptance of the inequal distribution of power which is interpreted as acceptance of hierarchy and the appreciation of high social status. Individuals with negative PDI values do not consider status as particularly relevant.

In the model a positive IDV indicates strong individualism, while a negative IDV represents a collective attitude. Individualists tend to join groups but are more likely to leave those again while collectivists strive towards a stable group membership.

Uncertainty avoidance (UAI) is modelled as the degree to which individuals are accepting changes with uncertain outcome both with regards to leaving groups 'for new challenges' as well as accepting new members in a group. Uncertainty avoidance is related to the homogeneity of an individual's or group's personal properties. Individuals with a positive UAI requesting a group membership dislike strong differences of a group's average personal characteristics to its own[8]. The UAI value is further coupled with the *experience* of an individual. Positive uncertainty avoidance correlates with a conservative attitude in case of bad past experience (such as being expelled from a group or a decline of group membership). Individuals with a negative UAI are comfortable to surround themselves with individuals inhibiting widely different characteristics and are rather encouraged by rejections and declines (i.e. have a somewhat deviant attitude).

Another crucial deviation from the original concept – motivated by the assumption that even Hofstede's cultural dimensions cannot be equally applied to each individual of a culture – is the random distribution of individually assigned cultural dimension values around the cultural dimension specified for the the society (i.e. centering the random distribution of individual assignments at UAI 0, PDI 0, IDV 0 for a neutral culture). Apart from the cultural dimensions their complement, the personal characteristics along with the social status, are integrated.

Personal characteristics are modelled as integers but should be interpreted as arbitrary symbols which express different values for the four given characteristics. The characteristics do not carry any specific semantic association (as mentioned above) and are of nominal scale. The differences of personal characteristics of an

[8]The operationalization of personal characteristics will be discussed in the next paragraph.

individual are thus defined as the sum of differing characteristics. Difference thus indicates the heterogeneity between two compared entities (which could be two individuals or a group (using the modes of all member characteristics[9]) and an individual).

Social status is modelled as integer values ranging from one to three indicating low, middle and high status.

The operationalized characteristics along with the basic agent behaviour allow to model an individual's or group's decisions. Functions are held semi-formal using PDI, IDV and UAI to represent the according values and mnemonic placeholders (format: <meaning of symbol>) for the other components. Additional inline descriptions for function components are provided. At this point first decisions taken by individuals are described.

Individuals decide if they want to engage in any group membership application process. Previous rejections in combination with the uncertainty avoidance values have negative impact on the interest of agents to join groups (*group interest function*).

```
/* group interest declines with rejections if individual is uncertainty-
 * averse; uncertainty-affine ones are even motivated by rejections
 * (which is the interpretation taken in this model) */
-1 * <number of rejections by groups> * UAI
//desperation of individual to belong to group encourages group interest
+ <desperation>
```

A positive result indicates interest to join groups.

After requesting group information from randomly selected groups, individuals rank the results in order to decide which group to join (by applying the following *group rank function* to all candidate groups).

```
/* Individualists favour groups of limited size to show more influence,
 * collectivists prefer large group sizes */
-1 * IDV * <group size>
/* if acceptance of power inequality is high,
 * the group status for the according group is taken into account */
+ (if (PDI > 0) {PDI * <group status>} else {0})
/* higher differences in non-cultural properties discourage risk-averse/
 * conservative individuals; risk-affine individuals favour differences */
+ -1 * UAI * <difference of non-cultural group characteristics>
```

[9] In case of multiple modes for average personal characteristics in groups one is randomly taken.

Higher values indicate a more interesting group.

Once having joined groups agents can deliberate whether they want to leave those groups using the following *group leave function*.

```
//High individualism motivates to leave group
IDV
/* if acceptance of unequal power distribution is high,
 * differences between own social status and group status
 * (group status is the mean of its member's status)
 * have according impact */
+ (if (PDI > 0) {PDI * (<own status> - <group status>)} else {0})
//a conservative attitude keeps individuals from changing groups
+ -1 * UAI
```

Values greater zero make the individual leave the current group.

From a group perspective complementary calculations apply to determine the handling of interactions with individuals. The representative for the group is the group leader. All requests to the members of the group (e.g. applications by individuals) are forwarded to the group leader for decisions. This way messages always reach an appropriate recipient.

The decision to accept an applying individual as group member is done via the *member acceptance function*.

```
/* if the group's mean individualism is low (i.e. collectivistic)
 * new members are desired */
(if (groupIDV < 0) {-1 * groupIDV} else {0})
/* if group is collective and applicant culture matches
 * group culture, use IDV in favour of applicant */
+ (if (groupIDV < 0) {if (<applicantCulture> == <groupCulture>)
  {-1 * groupIDV} else {0}} else {0})
/* if the group's mean power distance is high, the status difference
 * of applying individuals is taken into account (higher status of
 * applicant increases the chance for acceptance) */
+ (if (groupPDI > 0) {groupPDI * (<applicant status> - <group status>)}
    else {0})
/* uncertainty-avoiding groups (i.e. groups with high/positive groupUAI)
 * dislike agents which have either left or been rejected earlier;
 * uncertainty-accepting ones encourage dynamic memberships */
+ -1 * (<number of times the applicant has left this group>
    + <number of times he had been rejected by this group>) * groupUAI
/* uncertainty-avoiding groups avoid individuals with
 * strong non-cultural differences */
+ -1 * <difference of non-cultural characteristics (between this group
        and individual)> * groupUAI
/* if the group is uncertainty-accepting (negative UAI) and if culture
```

```
 * of applicant matches group culture (= average culture of members),
 * UAI is in favour of acceptance */
+ (if (groupUAI < 0) {if (<applicantCulture> == <groupCulture>)
  {-1 * groupUAI} else {0}} else {0})
```

As with most functions above, results greater zero indicate acceptance of the applicant, while values equalling zero or negative values result in rejection.
Another decision to be taken by groups (respectively their leaders) is the potential exclusion of members. This is done in two steps. First the most deviating member with regards to the perceivable non-cultural characteristics is identified. As a second step an *exclusion value* is calculated as following:

```
/* for uncertainty-avoiding groups the probability
 * of an exclusion of a member increases with
 * its deviance from the group's 'normal'
 * perceivable characteristics and vice versa */
groupUAI * <difference of non-cultural characteristics>
/* groups with high power distance take the status difference
 * between the individual and group into account */
+ -1 * groupPDI * (<individual status> - <group status>)
```

Again, values above zero lead to the exclusion of the member.
As pointed out in the high-level description the assignment of cultural properties is not done in a homogeneous manner. Societies are rather modelled by pursuing a random assignment of cultural dimension values with a distribution around the cultural mean values. This model thus assumes potential deviation from the mean cultural values – Hofstede's empirical evaluations themselves represent mean values of deviating individuals.
The independent variables for the experiments can thus be mainly reduced to the cultural dimensions, enriched with further tuning parameters (as described in subsection 6.4.1).
To measure the phenomenon of interest a considerable number of observations need to be taken ('Observations on the target' in Gilbert/Troitzsch framework). In consequence each round characteristics of individuals are recorded.
Those include:

- Individual characteristics (both personal as well as cultural (i.e. IDV, PDI and UAI)) which do not change once set.

- Group-related state – It indicates the state of the individual with regards to a group membership (not grouped vs. grouped). If the individual is member in a group, its group leader is of relevance as well.
- Number of group changes – The number of group changes an individual undergoes over time indicate the flexibility of an individual.
- Number of rounds since last group change – The number of rounds since the last group change indicates the stability of a group membership. Observed over the whole population it indicates a potential equilibrium in the demand to change group memberships (be it actively by leaving groups or passively by being expelled from groups).
- Number of times the individual has been rejected upon group membership request or expelled from groups.
- If the individual is a group leader (and thus represents a group), the cumulated number of individuals that have left the group is collected.

6.4 Implementation

6.4.1 General Aspects & Verification

Having discussed the model, aspects of its implementation should be reviewed at this point. As the model does not consider an explicit environment the implementation focuses on the 'cultural individuals'. All of those are implemented using Clojure as implementation language. Clojure scripts used for role implementations of communicating agents in μ^2 require the implementation of a `receive-msg [message]` function to which all received messages are passed (from the platform) in order to allow their processing in Clojure. They further require the implementation of a behaviour loop (`behave` function) which is triggered by the provided scheduler. Apart from this all relevant agent functionality (such as outbound messages according to the provided interaction patterns (e.g. `send-msg`)) as well as platform functionality (access to pseudo random number generator and scheduler) can be directly accessed from Clojure itself.

As the role implementations are potentially accessed synchronously – the messaging framework as well as the behaviour loop of the role itself – both threads

can operate on the same state. While the framework handles this via semaphores for Java implementations this is not necessary in Clojure as of its STM-backed transaction handling.

At this point it should be mentioned that only the behaviour loop of agents is scheduler-controlled, allowing (based on the number of initial requests and negotiations) various parallel and asynchronously executed conversations. As such only the 'motivation' of an agent is controlled in a fair manner – not his reactions.

In order to control all agents an additional so-called 'console-agent' is provided which allows to control the scheduler via command line. Along with this the console-agent is responsible for the collection of all statistical data. As agents in μ^2 are autonomous and do not allow direct introspection, at the end of each round every agent sends information about its current state to the console-agent which collects it and writes it to a flat file upon user request (via command line). Along with the real-time control aspect the interactive programming approach (and dynamic typing) of Clojure proves very convenient. It allows direct operation on the runtime state of the agent which is not only helpful to verify function output (and interactively correct the function) but also to accomplish debugging of appearing errors at runtime rather than replicating errors in order to trace their source (as with conventional Java).

Another model-related implementation aspect is the augmentation of all used cultural dimensions (UAI, PDI and IDV) with a weight factor to allow their fine-tuning respectively balancing at a later point. Experimenters can also define how many groups are contacted before deciding which group membership to apply for. Idea for this is to increase the 'rationality' of the group shaping process and limit effects of random selection of groups. Potential configurable parameters for the implemented model are thus:

- Culture definition(s) via UAI, PDI and IDV and number of agents for according culture
- Indicator whether society values should be randomized around the defined culture or strictly use the specified values[10]

[10]In fact only the 'randomized' version was used.

- Weights for all dimensions as well as rejections
- Number of rounds
- Number of requests to groups by individual agents before deciding which group membership to apply for

6.4.2 Validation & Sensitivity Analysis

While the verification aspect largely concentrates on the correction of syntactic and elementary functional aspects, the validation is crucial with regards to the expected overall model behaviour. At this point it should be recalled that the model is artificial and merely relies on the concept of the cultural dimensions for real-life grounding. No actual data on group formation processes is available to test it against; its orientation is rather explorative and yields towards an understanding of potential processes (in the trade-off of understanding and accuracy (see Axelrod [Axe97] as discussed in subsection 3.2.2)).

As distinct effects of the particular dimensions have been assumed during the modelling process (e.g. high status attracts individuals with high PDI values (see subsection 6.3)) this behaviour can be tested using a very small number of agents and the stepping functionality of the scheduler. As of the strong asynchronicity of the conversations, retracing full individual conversations of larger number of agents becomes practically infeasible.

The sensitivity analysis, and consequently the fine-tuning, have been undertaken in two steps. Initially all parameters are set to neutral values (all weights for cultural properties to 1, the mean culture to 0 (i.e. one neutral culture)) in order to gain a statistical baseline for the model's sensitivity towards parameter changes. Although simulation output is available for analysis μ^2 does not provide tools for analysis at the current stage. Thus a complementary post-processing workflow is created using the Konstanz Information Miner (KNIME) [KNI] which is a data mining tool built upon the Eclipse platform [Ecl] and serves as an open source alternative to commercial solutions such as the IBM SPSS Modeler [IBMa] or the SAS Enterprise Miner [SAS]. The workflow effectively filters the last round of all agents and analyses it both on grouped and ungrouped individuals in first

instance, drilling down into potentially interesting patterns.[11] Chosen indicators are *number of groups*, *average group size*, *quota of ungrouped individuals* as well as *average status and standard deviation* for in- and out-group agents as well as their average UAI, PDI and IDV values. In order to test the overall behaviour and reliability of results the different cultural dimensions are systematically adjusted ceteris paribus (i.e. 1st run: culture with neutral UAI, neutral PDI, neutral IDV; 2nd run: culture with high UAI, neutral PDI, neutral IDV; 3rd run: culture with low UAI, neutral PDI, neutral IDV, ...). In this setup all agents belong to the same culture; effects of group shaping *within* one, not *between* different cultures is tested. The target of the analysis is to balance the effect of all three cultural dimensions with regards to in-group/out-group fraction of all agents.

In order to determine the number of necessary rounds of execution to achieve clear and replicable results, pretests with an uni-cultural set of individuals (all individuals from one culture) were undertaken. Those revealed several clusters which will be introduced in detail in the result section 6.5. Stable groups, mainly caused by memorizing (and reacting on) drop-outs – from group side – or groups which declined earlier membership requests – by individuals –, developed after 25 to 100 rounds (depending on parameter set). Further clusters (e.g. frequently group-changing agents) within the out-group stabilized after about 125 rounds with minor exceptions during further runtime. Thus the sensitivity analysis was done using 200 individuals interacting over (generous) 300 rounds. To measure the reliability respectively tolerance of the outcome each configuration is executed five times, and the most centered results measured by in- and out-group fraction (i.e. number of grouped agents vs. number of non-grouped agents) were taken.

The results are shown in Appendix E.2 and its detailed findings are discussed in section 6.5 as those equally represent a result for this scenario beyond a pure sensitivity analysis.

It shows a balanced relation between agents which have joined a group and ones which did not. While PDI and IDV show a rather linear tendency driving agents to be inside or outside a group, UAI is modelled to be prohibitive for strong

[11] A screenshot of the iteratively elaborated stream can be found in Appendix E.1. The actual implementation can be obtained as indicated in Appendix F.2.

grouping as individuals tend to take risks in case of low uncertainty avoidance while high uncertainty avoidance keeps groups from accepting new (and considerably different) agents. In this context the aspect of weights for the different cultural dimensions is particularly relevant as those control the aforementioned effect. While an increasing weight for PDI and IDV positively correlates with the involvement of agents in groups, the UAI weight has a strong impact on out- and in-group fraction when testing the sensitivity of power distance[12]. Target is thus the balancing of in- and out-group fractions between PDI and IDV, as those fractions hardly changed when only adjusting UAI weights. To show this aspect results of the sensitivity simulation runs for UAI weight factors 1.5 as well as 1.8 are provided in Appendix E.2. In the following the weights 1.5 for UAI, 1 for both PDI and IDV and a weight of 0.1 on rejections (in order to limit the exclusion of agents from groups (see according functions in subsection 6.3)) have been chosen for all subsequent executions. In this setup agents request information from three groups prior to their decision which group to join. Desperate agents which have waited for more than five rounds consider the application to whatever response they have got without awaiting other outstanding responses.

The operationalization of the cultural dimensions turned out to be rather strict as the number of out-group agents is balanced around a level of 50 percent. Along with this the model implementation does not provide 'extreme results' but rather indicates trends. Average group sizes hardly reach three members, the maximum number of members reaches values up to ten. Potential reasons for this are threefold. Unless dissolved (by leaving members (group leader without members will dissolve their group)), groups will not give up their autonomy, thus not merge with other groups to shape larger groups. Another aspect is the green field approach. Groups are fully emerging from initial requests of agents. All agents are free to request immediately and are eventually only held back by high individualism (and wait for several rounds of 'increasing desperation' before requesting), leading to numerous rather small groups as of the equal distribution of initial agent requests. The third aspect is the agents' memory. Agents do not forget their interaction

[12] At this point it should be emphasized that a review of the code base has been made to ensure that there is no mistaken allocation of PDI values with UAI weight.

partners which is born out of the idea to make more 'intelligent' decisions on whom to contact and also to allow the emergence of stable relationships (past rejections make group leave unlikely (if individual seeks security (high UAI)) high number of rejections (e.g. based on low status of requestee) result in 'frustation' of individuals). This contributes to the rather limited average group size. Thus for analysis purposes size cannot be of sole importance.

Reviewing the structure of emerged groups two aspects demanded amendment of the concept: Groups did not show clear cultural patterns as expected initially. To reflect the idea that individuals might not be able to perceive the actual culture of the counterpart but are able to decide whether they share a cultural background which might attract to join a group, group leaders can distinguish if an applicant (for group membership) is of their own culture (in a boolean manner) and consider this accordingly. This mechanism weakens the modelling principle of avoiding the communication of any cultural properties – but not to the extent of 'knowing' each other's culture.[13]

Another aspect relates to the calculation of group properties. Initially cultural properties of a group are determined in a fair manner (i.e. average across all group members). This is amended by considering the properties of the group leader twice to amplify its influence within its group.

Given those amendments, experiments with different cultural setups are conducted to show patterns when combining different cultures in this model.

6.5 Results and Evaluation

6.5.1 Emergent Structures in Uni-Cultural Setup

As a first step – and in strong relation to the sensitivity analysis – experiments with a uni-cultural setup are conducted.[14] In this case all agents belong to a single culture; their cultural attributes are randomized around neutral cultural values (as described before). The configuration of a culture in this model shall be understood as *cultural coordinate* consisting of the values for UAI, PDI and IDV in the

[13]The operationalization description above already includes this feedback from the sensitivity analysis.

[14]To recall: The experiment was run with 200 individuals over 300 rounds.

listed order. The cultural coordinate for a neutral uni-cultural setup is thus [0;0;0]. As mentioned in the validation phase, characteristics of the out-group agents are analysed and show retraceable clusters which are described in more detail at this point.

One cluster of those out-group individuals consists of agents which frequently change their group membership as well as a cluster of agents which never joined any group as member themselves. Individuals of the latter cluster are of mixed status, often a higher degree of uncertainty avoidance and low power distance (i.e. those agents do not care about status). Their membership applications to groups are rejected throughout the simulation runtime. Those agents represent a 'permanent out-group'.

The cluster of agents which change their group memberships frequently can itself be subdivided into two subclusters. Agents whose status is above average and have an according high power distance (i.e. emphasize status) join groups (typically of lower status) but leave them, often within the next deliberation cycle, and thus – in the long run – remain as 'isolated elite'[15].

The other subcluster represents the largest cluster of out-group agents. Its members are characterized by low uncertainty avoidance (i.e. are tendentially more risk-taking), a slightly stronger focus on power distance but only an average status. As such members of this cluster can be described as 'gamblers'.

Grouped agents in turnaround do not show clear subclusters but can be contrasted against the out-group agents. Grouped agents have a lower status in comparison to the mean average of the out-group agents but a similar status to the gambling cluster. Their power distance is rather low, and in tendency they fear uncertainty. While this fear for uncertainty results in stable groups, the group formation process itself is driven by their collectivist orientation (i.e. negative IDV).

Depending on actual parameterization (i.e. culture coordinate) the results can deviate from the described patterns (e.g. elite status in collective culture is lower than others) and ratios between out-group clusters shift. However, for most cases the observations described are applicable. When reviewing the results in Appendix

[15] Again, 'elite' should not necessarily imply superior status but indicate their principal position, allowing them to opt out from group membership applications.

E.2 and retracing the patterns it should be emphasized that the mean cultural values can only be compared to other clusters of the same cultural coordinate (e.g. the PDI value of 1.8 (although seemingly of moderate level) can be considered low in an extremely status-oriented society). Table 6.1 summarizes the discovered patterns.

Interesting general findings from this uni-cultural experimental setup include that

Cluster	**Status**	**UAI**	**PDI**	**IDV**	**Group changes**
Grouped agents					
in-group agents	moderate	moderately uncertainty avoiding	moderate to low	collective	low (≈ 1)
Non-grouped agents					
Elite	moderate to high	high	high	indivi-dualistic	few
Gamblers	moderate[a]	low	moderate to high	mixed[b]	many
Permanent out-group	mixed[c]	high	low	mixed	none

[a] Status can strongly vary depending on cluster size (for different cultural setups).
[b] Gamblers show a slight tendency towards individualism.
[c] Status varies significantly with neutral status-orientation.

TABLE 6.1: Properties of clusters of individuals in uni-cultural setup

the mean status of in-group agents is nearly consistently lower than the one of out-group agents. As of the controlled group membership in-group agents have a more homogeneous status level (max. standard deviation of 0.6) while out-group agents' stat$\bar{u}$s are more widely spread (around 0.8) – independent from fraction between in- and out-group agents.

In collective, uncertainty-accepting and non-status-oriented societies the fraction of 'gamblers' is considerably high. While the first two characteristics encourage independence and are mainly decided by the individuals, strong status-orientation has a similar effect but is a consequence of the groups' decisions to reject individuals.

Summing up the effects of the different dimensions with regards to the group shaping mechanism in this model collectivism as a key driver to shape groups while

power distance controls potential new members[16]. Uncertainty avoidance determines the group stability in the long run (high uncertainty avoidance of group members can limit group growth but makes agents remain in groups).
Important to mention again is the fact that although group sizes and stat$\bar{u}$s only show limited variation between the different clusters, the results are reproducible; limited variations do not change the semantic implication.

6.5.2 Multi-Cultural Experiment

For further experiments – and to test the actual culture feature – a multi-cultural setup is considered using the combination of various synthetic culture definitions. Apart from a neutral culture (coordinate [0;0;0]) eight further cultures, exploiting all possible extreme combinations, have been defined as shown in table 6.2. Those

Cultural coordinate	**Description of synthetic culture**
[0; 0; 0]	neutral baseline culture
[4;-4; 4]	uncertainty-avoiding, not status oriented, individualistic
[-4; 4;-4]	uncertainty-accepting, status-oriented, collective
[4;-4;-4]	uncertainty-avoiding, not status-oriented, collective
[4; 4;-4]	uncertainty-avoiding, status-oriented, collective
[-4; 4; 4]	uncertainty-accepting, status-oriented, individualistic
[-4;-4; 4]	uncertainty-accepting, not status-oriented, individualistic
[-4;-4;-4]	uncertainty-accepting, not status-oriented, collective
[4; 4; 4]	uncertainty-avoiding, status-oriented, individualistic

TABLE 6.2: Combinations of Synthetic Culture Coordinates

definitions of the cultural coordinates do not use the full scale of the model specification (i.e. values of 5 and -5) to improve the randomization around the extreme values.
For the multi-cultural setup the objectives are changed from the view on emerging group clusters towards a stronger focus on the performance of the individual synthetic cultures within emerging mixed groups. Consequently each culture is equally represented in the setup. As the JVM memory limitation on the test machine – on 32-bit Windows XP this is about 1.5 GB[17] – currently allows to run

[16]Status-oriented groups do not necessarily yield in higher status but rather limited number of groups.
[17]This is the case despite more available RAM. For an explanation on this refer to [Bru].

about 650 (cultural) agents[18], the number of agents for individual cultures is set to 72, resulting in 648 agents. Individuals interact over 300 rounds. As before the differences in results for different runs are neglectable for the perspective taken here. Representative results are shown in Appendix E.3 and analyzed in the following.

For this analysis the cultural components of the resulting groups are of particular interest. All cultures are represented in the group shaping process – but with widely differing outcome. When observing the in- and out-group fractions (as shown in Figure 6.1) those vary significantly across different culture setups. No particular trend respectively unidimensional dependency of group size can be observed. More interesting than group size is in fact the differences between the representation of cultures as group members vs. group leaders. Thus a closer look

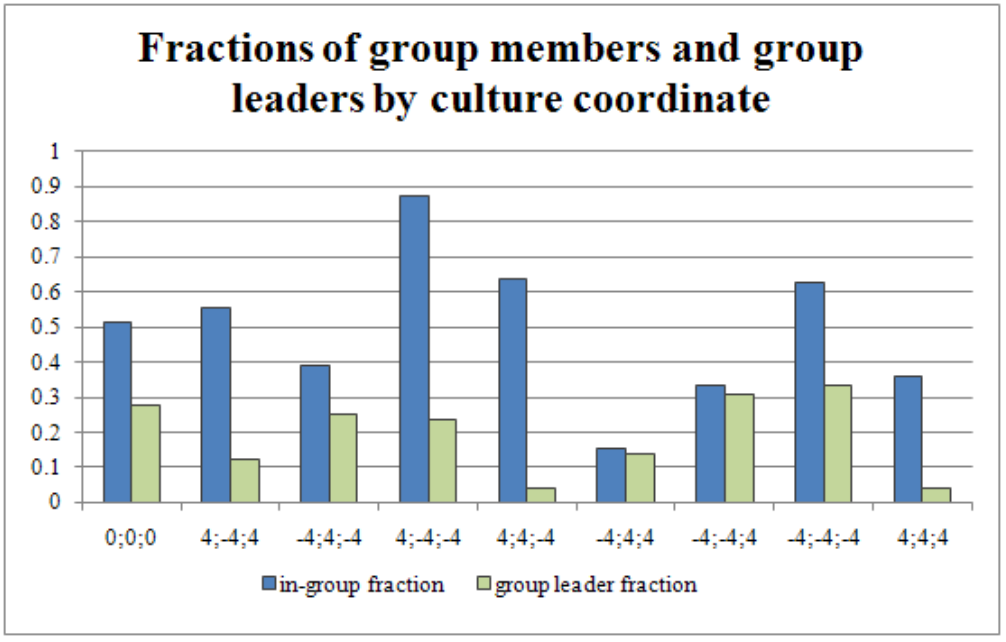


FIGURE 6.1: In- and out-group fractions in the multi-cultural simulation setup

at group structures with focus on cultural distribution of group leaders is of interest (see charts in Figure 6.2). While the distribution of group memberships for all synthetic cultures is considerably balanced (apart from outliers such as coordinate [-4;4;4], both being uncertainty-accepting and individualistic), leaders of those groups mostly belong to cultures with particularly 'inviting' characteristics.

[18]This limitation is caused by the larger footprint as a consequence of the Clojure implementation of the individual agents.

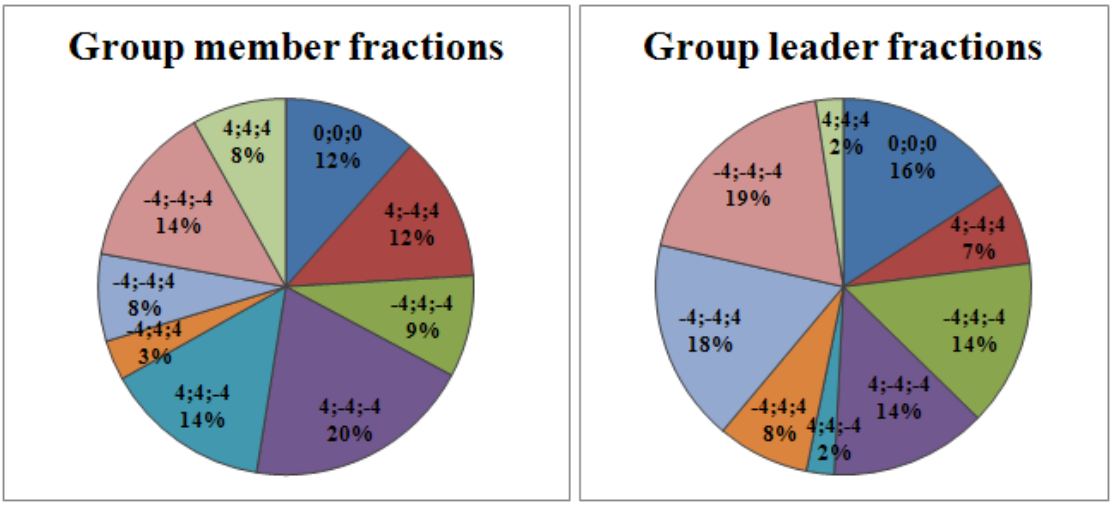


FIGURE 6.2: Groups by member culture and leader culture

In this model leaders derive from mostly uncertainty-accepting groups as those are more likely to accept unknown new members. This behaviour also clarifies the different understanding of individualism and uncertainty-acceptance in the model. Here uncertainty-acceptance has a more passive effect of tolerating others' wishes/requests which eventually encourages the shaping of new groups – in fact an effect rather similar to collectivistic agents. Individualism, in contrast, describes the 'motivation' of an agent to actively pursue its own goals (such as joining or leaving a group). As such UAI shows a stronger effect in groups while IDV is of stronger effect for individuals – in this model.

The fractions of group-leading agents in fact need to be evaluated in the context of the in-group ratio for a specific culture. An example culture is [-4;4;4] which has a limited involvement in group memberships ($\approx 3\%$) – about 85% of individuals from this culture are not grouped (see Figure 6.1) – and a relatively higher importance with regards to group leaders ($\approx 8\%$). In fact nearly all agents of this culture are group leaders (group leaders are group members at the same time), as indicated by a leader/member ratio of 0.9 for this culture. According ratios for all cultures are shown in Figure 6.3, giving a clear picture on the relative performance of different cultures with regards to group leadership – independent from in-group ratios for specific cultures. Aggregating those results by coordinate components, as shown in Figure 6.4, clarifies the relative importance of all cultural dimensions

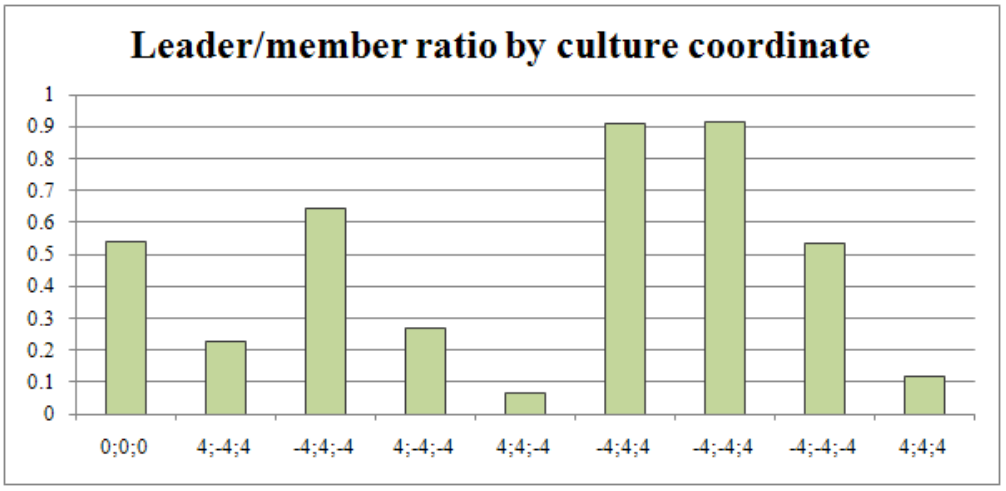


FIGURE 6.3: Leader/member ratio of in-group agents by culture

for the group shaping process and in particular the ratio between group membership and group leadership.

The figure in fact manifests the suggestion that the UAI dimension is of central

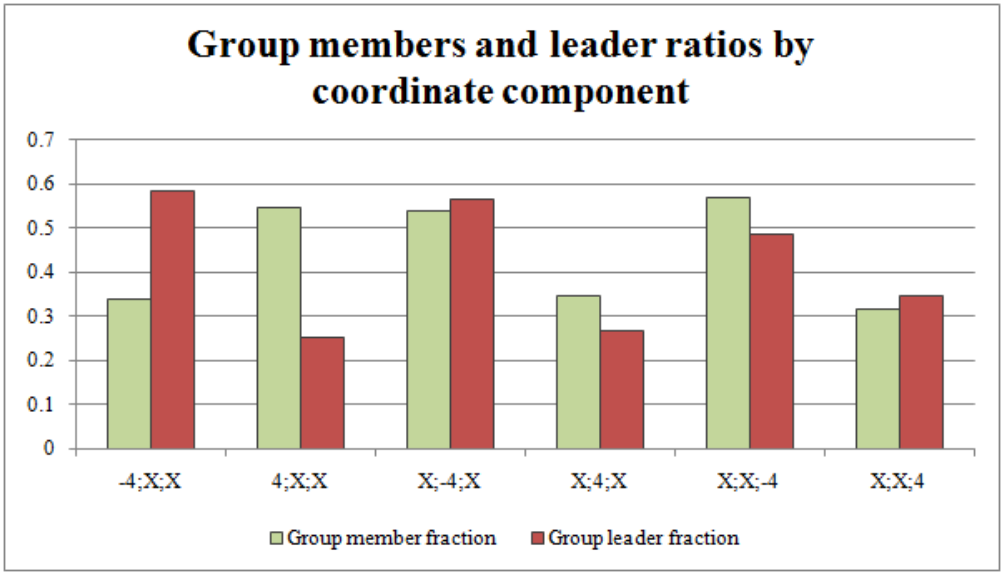


FIGURE 6.4: Leader/member ratio by culture coordinate component

relevance when determining cultures which are more successful in providing group leaders. This effect is retraceable when considering the openness of uncertainty-accepting groups to new members while uncertainty-avoiding ones are restrictive with regards to newcomers. Agents with high uncertainty avoidance ([4;X;X]) (and a higher drive to be in stable group relations) have a reasonable chance to become group members with a poor chance of being leaders of those groups while

individuals with low uncertainty avoidance ([-4;X;X]) do not spend long periods of time in a pure member group relation (especially in combination with high individualism). However, if they are asked for group information they are very likely to start a group, accept new members and remain as group leaders in the long run which explains the significant shift between group membership and leadership fractions for different cultures.

In contrast to the uni-cultural setup which shows large differences for in- and out-group fractions for extreme UAI values, group membership fractions in the multi-cultural model are in fact nearly balanced for all dimensions as the multi-cultural setup allows the differentiation between member and leader cultures, providing a more fine-grained picture on factors relevant for the group shaping – respectively an answer to the question "Who is starting groups?".

The other dimensions do not show this inverse correlation of probability for group membership and leadership. In the case of PDI it can be briefly said that low power distance agents ([X;-4;X]) are likely to end up in groups and are likely to become leaders themselves (more than 50%). In this model individuals with strong status-orientation ([X;4;X]) are not only less likely to be in groups but are also hardly leaders. Here the reason is that status-orientation does not imply that all individuals necessarily have a high status but rather segregates individuals by status. If they have a high status along with status-orientation they are likely to opt out from groups soon after entering and remain outside (as indicated before for the 'elite' out-group cluster in the uni-cultural setup) or show strong group changing behaviour when in conjunction with other dimensions (e.g. individualism) (such as the cluster of 'gamblers'). Both agents belonging to the 'elite' as well as the cluster of 'gamblers', the major out-group clusters, are likely to show a strong status-orientation. A consequence from this is that the average status of groups in status-orientated cultures is rather low; high power distance additionally limits group size (see according tables in Appendix E.2).

Members of collectivistic cultures ([X;X;-4]) are likely to be part of groups and have a reasonable high chance (though less then 50% for this model) to be group leaders. Members of individualistic cultures ([X;X;4]) are consequently less likely

to be part of groups in the long run (about 30%). However, their chance of being leaders of those groups ranges at around about one third. This result is retraceable when considering that individualists find a stronger representation in the out-group clusters with higher likeliness to switch groups frequently. Unless combined with low uncertainty avoidance they are unlikely to form new groups upon requests by other agents.

In order to allow a third party to retrace the modelling decisions and to reproduce the results[19], the model code itself can be retrieved as described in Appendix F.2.

6.6 Summary

Overall the model of group shaping by means of cultural characteristics described here shows an intermediate approach to consider 'culture' in agent behaviour. Apart from the unexpected result with regards to uncertainty avoidance and potential group leadership (for this model) it allows a loose orientation on cultural dimensions empirically elaborated by Hofstede. Consequently this allows to define agent societies using actual cultures as archetypes. However, doing so the experimenter should be aware of the shortcomings of this model.

Limitations of the current model include the rather unidimensional interpretation of cultural dimensions. For example, PDI is more or less simply interpreted as status-orientation, leaving out reactive aspects of the subaltern upon actions of the group leader and further characteristics suggested in the literature (see Hofstede [Hof01], p.108 for more examples). As such the model takes a rather naïve approach to model the dimensions.

Another limitation is the yet only static-comparative analysis. Introspection at runtime is feasible but work-intensive. Primitive mechanisms to visualize historic developments of groups are available[20]. However, those merely concentrate on group sizes but leave out the other, in fact more relevant aspects as revealed using

[19]Recalling raised demands for this from the literature (see subsection 3.2.4): Heath et al. [HHC09] demand model code in order to ensure reproducibility of results, Michel [MGF04] describes the engineering divergence phenomenon for reimplementations based on even complete specifications. Troitzsch [Tro04] sees the implementation (including simulator) itself as a 'structural reconstruction' of the theory. Code provision thus complements the incomplete description provided to this point.

[20]For this purpose the Clojure-based statistics/visualization package Incanter [Inc] has been used.

KNIME for data post-processing[21].

Along with this, and less related to the simulation application but the platform as a whole, the limited visualization capabilities need to be mentioned. Future work on μ^2 will particularly target this concern. Although the platform provides powerful runtime facilities, it is hardly usable for non-programmers but allows a quick start for developers being either familiar with LISP/Clojure or Java. For a decision which language to use two factors should be taken into account. Code written in Clojure is likely to be significantly more compact. A parallel implementation in Java – done for the early iterations of the simulation scenario – revealed the need for about twice as many lines of code in comparison to Clojure. The advantage of Java code is the – though not quantified at this point – lower memory footprint of applications. In order to execute larger numbers of agents (the current simulation scenario is limited to about 650 agents), Java is yet the language of choice.

Summing up, the current scenario implementation is set up to meet a trade-off of simplicity (by avoiding strong reasoning elements) and expressiveness/usefulness to allow the use with considerably large numbers of agents[22]. Future model refinement can include the modelling of merging groups with multiple levels of organisation or the simulation of impacts on different cultural societies (e.g. external: aggressor culture, internal: dictator) on emerged groups. Of particular interest for this is the application of stability measures for different (e.g. collective or individualistic) societies taken from systems theory. Additionally the other cultural dimensions (Masculinity and Long-Term Orientation) could also be integrated.

Beyond that the model can serve as one approach to consider culture characteristics for agent in non-simulation MAS. For MAS in general the model allows the representation of culturally augmented behaviour for individual agents, such as a culture-based strategy selection in the case of agent cooperation (degree of risks; matters of harmony in collective agent societies (keeping one's 'face')) or coordination (e.g. auction behaviour).

[21] As the data are provided in a tabular (CSV-like) manner, processing with other tools such as spreadsheet programmes is equally possible.

[22] This should be seen in contrast to other rather high-level modelling approaches such as [HJV09].

Chapter 7

Conclusion

In this chapter we will provide a final discussion on the outcomes of this work, together with the shortcomings and outlook of future research aspects.

7.1 Summary of Achieved Objectives

This thesis describes a wide range of aspects and yields towards comprehensive, extensible and light-weight solution integrating agent-based modelling with modern technologies.

To achieve this work retraces the wide range of definitions and imprecise agent understanding within the MAS community alone – without initial consideration of the simulation community. One outcome of this is the fact that the attribution of agency might be possible to virtually any computational unit. Other views adocate agents as standard-compliant reasoning-centric individualists. To break up the stereotypes suggested by the different definitions, this work suggests a dynamic notion of agency. Different notions are applicable to particular levels in to specify a context-related agent understanding and serve as communication vehicle throughout various parts of the thesis. Apart from this dynamic notions provide potential to capture future agent capabilities but demand for a more extensive elaboration than done in this work.

As a next step this work retraces characteristics of multi-agent systems and relates those to the system theory as a potential blueprint for any kind of (complex)

system. The explicit consideration of system-related elements (such as the environment) makes multi-agent systems attractive for a considerably close interpretation of the system theory. However, the work also highlights an efficiency-related trade-off of a too 'direct' interpretation with regards to the consequent separation of different system levels.

Beyond the conceptual foundations this work focuses on two research fields, Agent-Oriented Software Engineering and Agent-Based Social Simulation, both of which are related by their use of agents as modelling tool. Similar to the field of MAS alone, those two fields have a significantly differing agent understanding. AOSE has the tendency to support comparatively strong notions of agency while social simulation favours weak notions which strongly rely on the ascription of intentionality rather than its actual incorporation. Analysing the differences en detail, i.e. the platform level, simulation puts a strong emphasis on fair scheduling, indirect communication – along with a stronger importance of the environment – and visualization respectively reporting/analysis features while AOSE platforms focus on the agent concept itself and expressive direct communication mechanisms along with advanced features such as conversation management. Although the micro-agent concept advocated by this work originates from the field of AOSE it in fact resides at the intersection of both approaches, complying to AOSE in its fundamentals while meeting the tendency of simulations to integrate more powerful agent modelling and communication features.

With relation to the relevance for agent-based systems the thesis includes a brief introduction on concurrency respectively concurrency handling mechanisms in order to structure the approaches of selected candidate technologies for a reimplementation of the micro-agent layer. Two of the technologies, Clojure and Asynchronous Message Passing for Java, are ultimately tied to the redesign of the framework while Android's capabilities suggest an adaption of the micro-agent framework with particular features targeting this mobile application platform.

The work brings the widely differing aspects of the earlier parts together and introduces the micro-agent concept along with its current implementation. The limitations of the legacy framework build the basis to develop requirements for a

reimplemented micro-agent framework. The redesign suggests significant extensions to the original concept, such as full network support (respectively the abstraction of applications from it), the fully encapsulated handling of concurrency as well as the introduction of Clojure as powerful alternative agent implementation language. For an alternative use with social simulations the use of Clojure's STM provides a powerful tool to model and coordinate agent access to shared environments. Additional to this a fair scheduler, enforcing round-level fairness is provided for optional use. Beyond those functional aspects the platform now provides a further modelling mechanism which not only abandons a code-intensive modelling of sub-agent relations of its agent organisations but also offers compliance with the system understanding suggested in the system theory. This gives developers a pragmatic level of decisional freedom to favour convenience and a level of modelling consistent with system theory. Alternatively, developers can design a purely efficiency-oriented agent interaction without any restriction by system-theoretical abstractions.

Overall, the reimplemented framework now has powerful management mechanisms via its multi-level approach of primary group relations and expressive communication modelling mechanisms. In consequence it takes a generational step from a considerably pure meta-model implementation towards a fully-fledged self-contained multi-agent, or better, micro-agent platform. As the legacy framework lacked numerous management features along with the concurrency concerns, its primary role was to provide the core for the more powerful high-level OPAL agents. This new platform, its improved feature set and usability (concurrency handling, network, communication patterns, 'tuning-by-configuration') along with high performance, inverts this relation. Consequently the use of OPAL with its standard-oriented communication capabilities is now optional. Application developers need to decide whether they need standard-compliance when choosing OPAL rather than avoiding the limited feature set on micro-agent level. However, even when OPAL communication is needed, the communication capabilities of μ^2 allow the parallel use of both platforms, e.g. to communicate with low-level services such as web services or making Android functionality accessible to OPAL agents (e.g. sending

SMS via reasoning agent). As such the reimplemented framework opens a range of new functionality not only for micro-agents but even for coarse-grained standard-compliant agents building on top of it, making OPAL a multi-level multi-channel agent platform.

Another aspect is the provided Android port of the platform. It opens a potential playground for future research when considering the direct interaction of micro-agents and application platform (i.e. 'operating system' from the agent point of view), providing an open environment to micro-agents. Micro-agents show a particular suitability for exactly this purpose as they are not confined to communication 'with their own kind' but access low level information sources and have a low memory footprint, allowing them to run on virtually any platform. The concept of agents naturally incorporates the idea to act in open systems which makes this approach interesting beyond the pure implementation of mobile applications. From the latter perspective however, μ^2 , respectively MOA, can act as a middleware bridging desktop and mobile world, not only for agent applications but basically any application requiring message-based interaction between an arbitrary number of stationary or mobile platforms.

The last aspect, serving as a proof of concept – apart from simple benchmarks to test and measure the feasibility of earlier aspects – is the modelling of a simulation scenario which uses the communication mechanisms of the platform, its scheduling mechanism and the ability to implement agents in Clojure. The concept itself relies on the empirically backed Cultural Dimensions by Hofstede to model cultural individuals and analyze their group shaping behaviour. Its goal is to provide a potential understanding how those dimensions can be used to augment agents with cultural aspects, similar to human societies. Particular outcomes are twofold. For uni-cultural societies, set up in the context of a sensitivity analysis, three out-group clusters consistently emerge for any configuration, and in fact clarify why agents eventually *do not* end up in groups. The multi-cultural setup, combining all different extremes of cultural dimensions, provides an interesting insight into grouped agents. Particularly the differing composition of groups with regards to their predominant member culture in contrast to the dominant leader culture is

interesting. It offers suggestions why some cultural characteristics might be more likely to produce followers while others make individuals more likely to abstain from explicit group shaping or enter groups only as leader.

With regards to the execution of the simulation the platform shows both a reliable (with regards to result replication) as well as robust behaviour (with regarding to the handling of massive numbers of concurrent interactions). Overall, the platform largely satisfies the expectations and is a profound base for future developments – be it in the area of AOSE or ABSS; its development has opened various directions of potential future research.

7.2 Limitations and Future Work

Along with the achievements of this work, various drawbacks are equally noteworthy and are discussed in conjunction with an outlook on future work.

The design and implementation of the micro-agent platform μ^2 has reached a complete status but yet seeks for intensified use by further applications. One available extension for the new micro-agent platform is a Coloured-Petri-Net-based conversation manager which allows the explicit modelling of complex conversations with optional externalization of agent functionality.

Current limitations of the platform include generic mechanisms to communicate with web services which yet demands for a strong intervention by the application developer. It should be equally simple as the use of the agent decomposition facilities.

Another aspect is the lack of integrated reasoning mechanisms to get closer to scale up to any (conceptual) 'weight' of agency. Candidates for this purpose are JavaPRS (jPRS) which had been optionally provided with the original micro-agent framework as well as the rule-driven ROK (both to be found under [Javb]). Apart from those the integration of recent approaches incorporating the de facto standard for declarative agent implementation, AgentSpeak, such as the Jason reasoning engine is considerable. New challenge arising from this is the amendment of reasoning engines themselves to access lower level agent functionality (e.g. extension of language primitives to address non-reasoning micro-agents).

To ease the integration of additional functionality by application developers a

dedicated plug-in mechanism is desirable. This feature is planned as future work. Currently only option (apart from actually modifying the platform code) is to use the event subscription mechanism to subscribe additional micro-agents (which encapsulate the required functionality) to according (platform) events.

Further the use of Clojure provides the foundation to provide mobile agents. Agents developed in Clojure could be easily serialized by explicit analysis of its namespaces and – at least in the sense of weak mobility (see subsection 2.1.1) – sent over the network and reinstantiated remotely. As a subset of this, Clojure-based agent roles can already send code to each other and directly execute it, allowing to model a notion of *learning*. Agents can simply forward their capabilities (in the shape of functions) to other agents which can incorporate it. Thinking this aspect further, Clojure is utmost suitable to be basis for reasoning engines itself, particularly as of its concise S-expression-based syntax which is in fact similar to syntax for FIPA content languages. This allows a homogeneous implementation of both knowledge representation as well as agent communication language on the platform.

Limitations for the use of Clojure yet includes the lacking support on Android. The Android port of the micro-agent platform does not yet allow the execution of Clojure-based role implementations. Apart from this (yet) Android-related downside the implementation of MOA should be elaborated further to provide more Micro-agent/Android interaction patterns and also include considerations to automatically identify mobile versions of the platform and provide its capabilities to stationary agents in an automated fashion (e.g. communication channels, internet access, calendar access for synchronization or reasoning to schedule appointments). The realm of applications for the blended use of micro-agents and Android is wide. It ranges from simple 'intelligent' applications with low memory footprint to robotics or the use of MOA as middleware for meaningful interoperation of Android devices or simply to allow access to Android functionality from desktop devices, driving the convergence between those further.

In the context of simulations the key drawback of the current implementation is

the lack of analysis and reporting features. While the platform provides powerful mechanisms for agent modelling as well as in/direct communication and fair execution, the reimplementation does not satisfy this requirement at the current stage. Means to analyze execution results are yet limited to the output of flat files which demand for further introspection using external tools. Future versions should provide this out of the box as a simulation-related extension to make the platform a potentially attractive alternative to existing simulation systems. Beyond the pure analysis aspects this should also include real-time visualization as well as real-time control of the scheduler – yet the scheduler can step (i.e. take one round) or run for a defined number of rounds/time but cannot be interactively stopped or started.

Visualization is not only a drawback affecting the simulation perspective; for the purpose of AOSE the platform should additionally be augmented with a graphical modelling mechanism to allow a more explicit modelling of the agent organisation, μ^2's striking feature. Structural code could thus be automatically generated; the developer could concentrate on the pure agent internals. This approach would not only provide an advantage when modelling applications but also for their analysis as part of their maintenance. An expressive visualization of the agent organisation provides a documentation mechanism as part of the platform, similar to an UML class diagram in the case of object-oriented programming. In conjunction with the conversation manager mentioned above this would allow a full static analysis of both agent organisation and conversation protocols out of the box which – as a further step – is the basis for an automated documentation of dynamic agent behaviour. When combining those aspects the platform would in fact be self-documenting and offer a better understanding of the non-deterministic behaviour (especially with regards to debugging).

Besides those visions for future development, the micro-agent framework is unlikely to allow a fully graphical modelling, even in the long run. As micro-agents particularly target heterogeneous low-level functionality and yield for efficient execution, their implementation will still demand for knowledge of according implementation languages as well as functional principles of integrated external resources (such

as the case for the MOA-mediated interoperation between micro-agents and Android).

Concluding, micro-agents are a pragmatic approach to combine the modelling principles of AOSE with performance. The reimplemented version of the micro-agent platform additional adds flexibility as a further ingredient. Micro-agents can access heterogeneous resources and act as a middleware for distributed applications. This, combined with high execution performance and an integration-friendly approach to new technologies makes micro-agents appealing for the use in mainstream software engineering. Beyond that, their small-scale nature supports the exploitation of emergence which – paired with expressive direct communication – makes micro-agents attractive for application fields beyond the realm of AOSE.

Appendix A

Listings and Class Diagrams of μ^2

A.1 Pseudo-code for Dynamic Binding Mechanism

```
1  if(validationActive){
2     - validate message
3     if(invalidMessage){
4       - notify sender that message is invalid
5       - stop further processing at this point
6     }
7   }
8   if(recipientIsDefined){
9     - pass message to Message Transport Layer (MTL)
10  } else {
11    if(messageContainsIntent){
12      - look up first recipient who plays role which can satisfy this intent
13      if(recipientFound){
14        - set recipient in message and pass to MTL
15      } else {
16        if(otherPlatformsConnected){
17          - send message to other platform and notify sender about this (either
18            found agent on other platform will reply or remote platform itself
19            will notify on failure to find agent for intent)
20        } else {
21          - notify sender that no agent could be found to satisfy intent
22        }
23      }
24    } else {
25      if(messageContainsEvent){
26        - check on event subscription change (add/remove)
27        if(subscriptionRelated){
28          - handle subscription management
```

```
29        } else {
30          - assume raised event and dispatch to all subscribers (and depending on
31            event parameter also to other nodes) by passing to MTL
32        }
33      } else {
34        - notify sender that message is invalid (neither recipient, intent nor
35            event defined) - will never be called if message validation is activate
36      }
37    }
38  }
```

Listing A.1: Pseudo-code for Dynamic Binding Algorithm

A.2 Examples for Micro-agent usage in μ^2

A.2.1 Micro-agent Interaction Example in μ^2

This interaction example 'translates' the scenario as outlined in the subsection 5.1.1 (Listings 5.1 to 5.3) to the new micro-agent platform.

```
1  public class ServiceProvider extends DefaultSocialRole {
2
3    public void initialize(){
4      addApplicableIntent(ServiceIntent.class);
5    }
6
7    public void handleMessage(MicroMessage message){
8      if(message.getIntent().getClass().equals(ServiceIntent.class)){
9        MicroMessage msg = message.createReply();
10       msg.setPerformative("COMMIT");
11       send(msg);
12     } else {
13       MicroMessage msg = message.createReply();
14       msg.setPerformative("NOT_UNDERSTOOD");
15       send(msg);
16     }
17   }
18
19   public void release(){}
20 }
```

Listing A.2: Implementation of ServiceProvider in μ^2

```
1  public class ServiceCustomer extends DefaultSocialRole {
2
3    public initialize(){}
4
5    public void handleMessage(MicroMessage message){
6      print("Goal result: " + message.getIntent().toString());
7    }
8
9    public void release(){}
10
11   //helper method defined to initiate interaction
12   public void startInteraction(){
13         //send intent
14         send(new ServiceIntent());
15   }
16 }
```

Listing A.3: Implementation of ServiceCustomer in μ^2

```
1 public static void main(String[] args){
2   //initialize agent with according role
3   SystemAgentLoader.newAgent(new ServiceProvider());
4   ServiceCustomer customer = new ServiceCustomer();
5   SystemAgentLoader.newAgent(customer);
6   customer.startInteraction();
7 }
```

LISTING A.4: Main method to start interaction in μ^2

A.2.2 Usage of MessageFilter in μ^2

```
1  public class DecompositionExampleRole extends DefaultSocialRole {
2
3    protected void initialize() {
4      //definition of pattern
5      MicroMessage pattern = new MicroMessage();
6      pattern.setSender("OtherAgent");
7      pattern.setPerformative("REQUEST");
8
9      //adding of message filter with inline implementation
10     addMessageFilter(new DefaultMessageFilter(pattern) {
11
12       protected void initializeMessageFilter() {
13         /* potential further initialization code
14          * (e.g. another message filter level (cascading))
15          */
16       }
17
18       public void onMatchSuccess(MicroMessage message) {
19         //action upon successful match
20         print("Received REQUEST message from OtherAgent");
21       }
22
23       public void onMatchFail(MicroMessage message) {
24         //action upon failed match
25         print("Received message from different agent: " + message.getSender());
26       }
27
28       public void release() {
29         //execute any action upon before death of agent (i.e. closing resources)
30       }
31     });
32   }
33
34
35
```

```
36    public void handleMessage(MicroMessage message) {
37      /* Handling messages is unnecessary if all
38       * functionality is delegated to message filters
39       */
40    }
41
42    public void release() {
43          /* eventual user-defined actions when shutting down
44           * super-agent (message filters will be stopped
45           * automatically)
46           */
47    }
48
49  }
```

LISTING A.5: Initialization of MessageFilter in μ^2

A.3 Class Diagrams of Platform

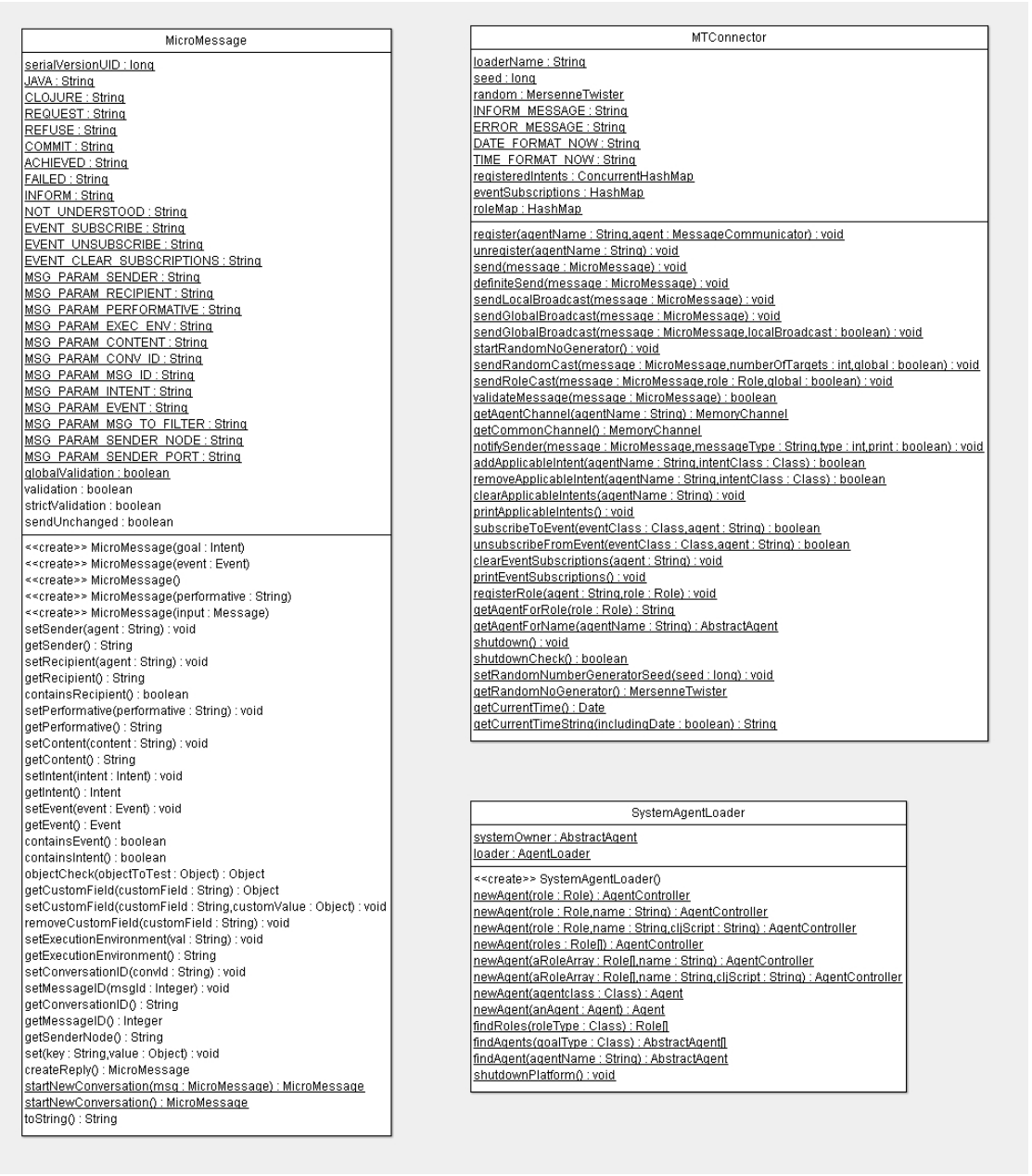


FIGURE A.1: MicroMessage, SystemAgentLoader and Message Transport Connector

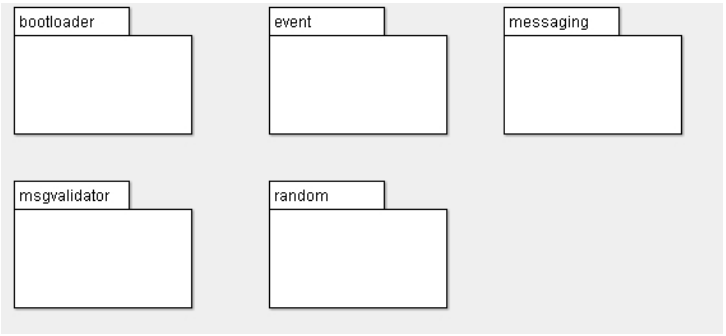


FIGURE A.2: Sub-namespaces of org.nzdis.micro

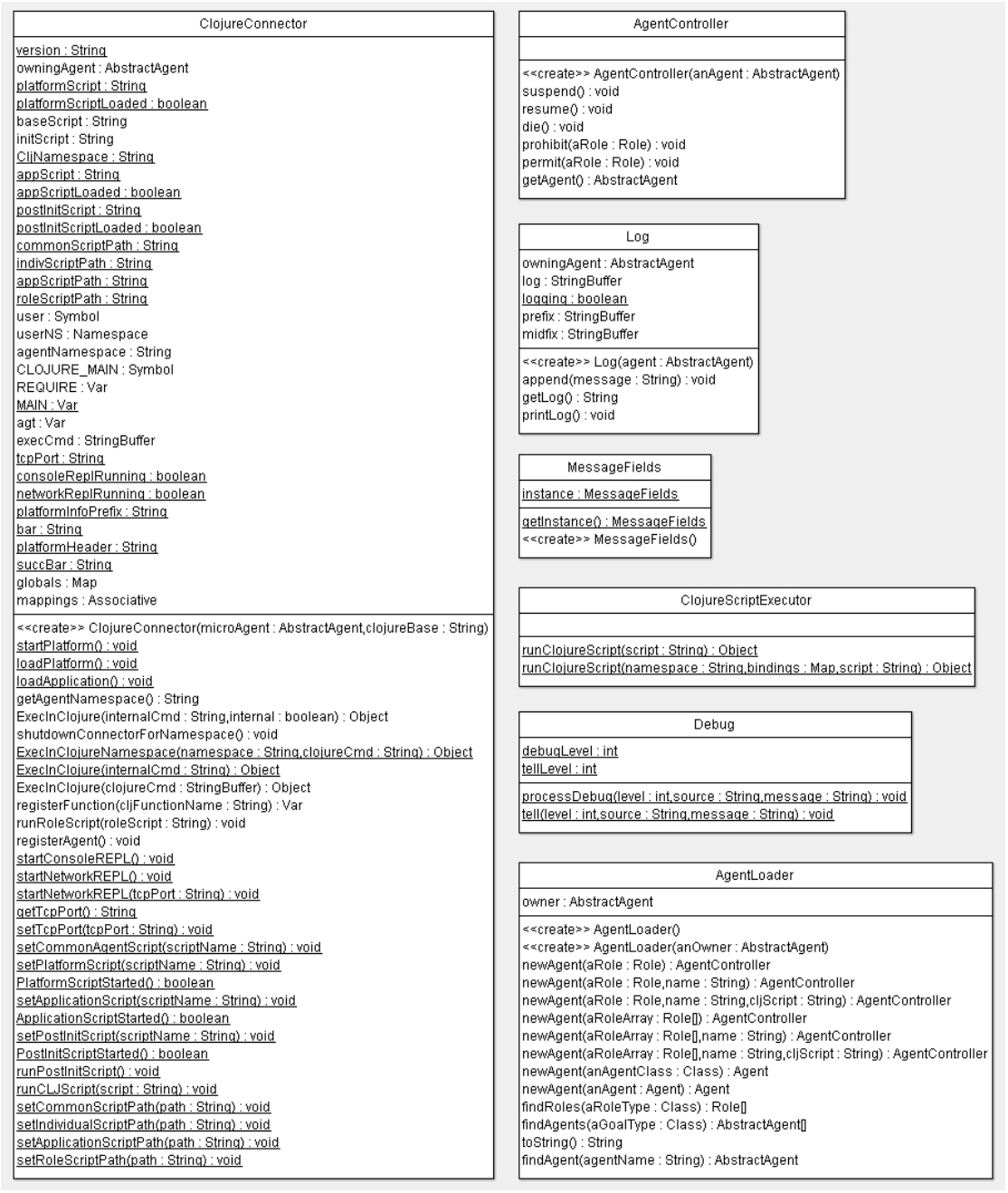


FIGURE A.3: ClojureConnector and further classes

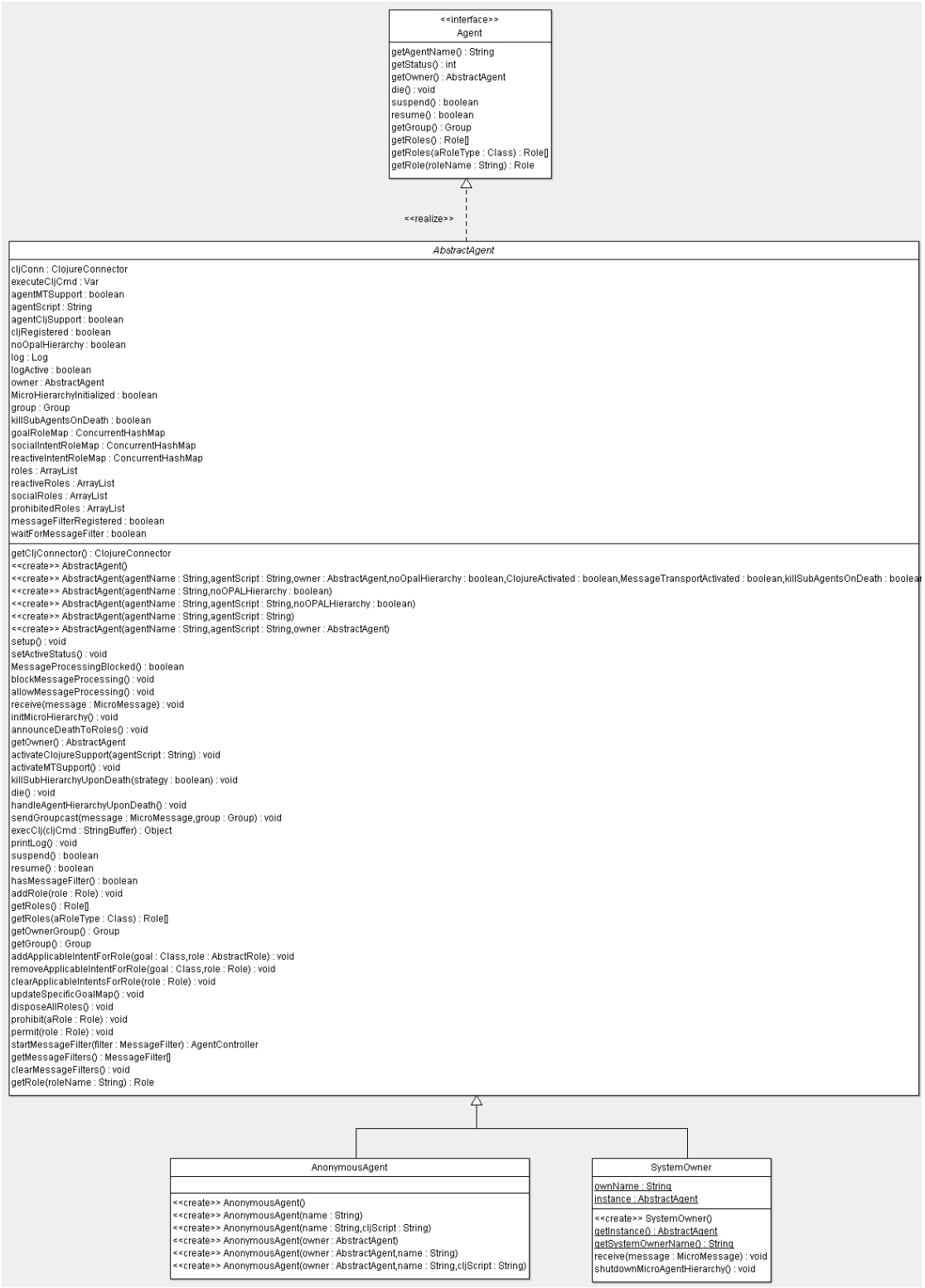


FIGURE A.4: AbstractAgent class

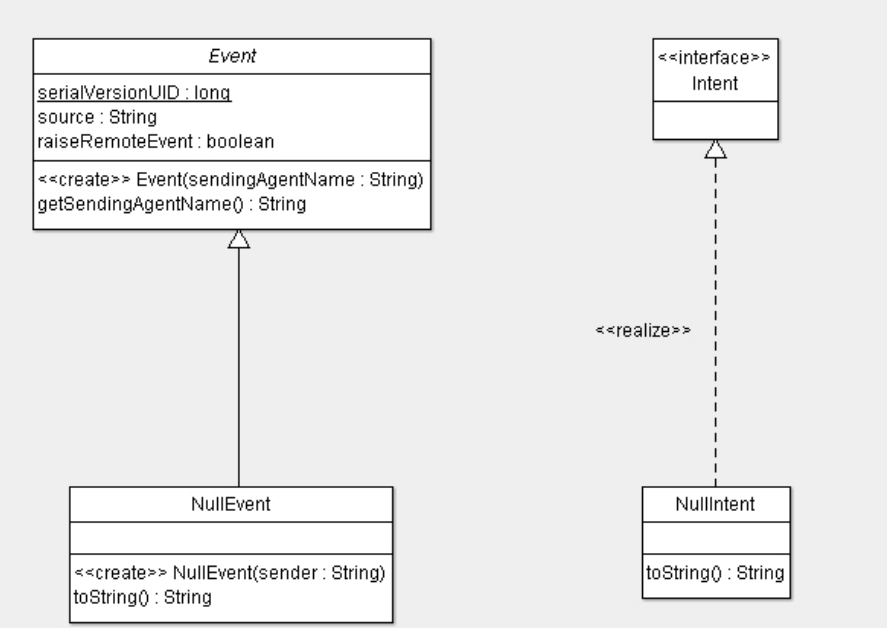


FIGURE A.5: Event and Intent interfaces/classes

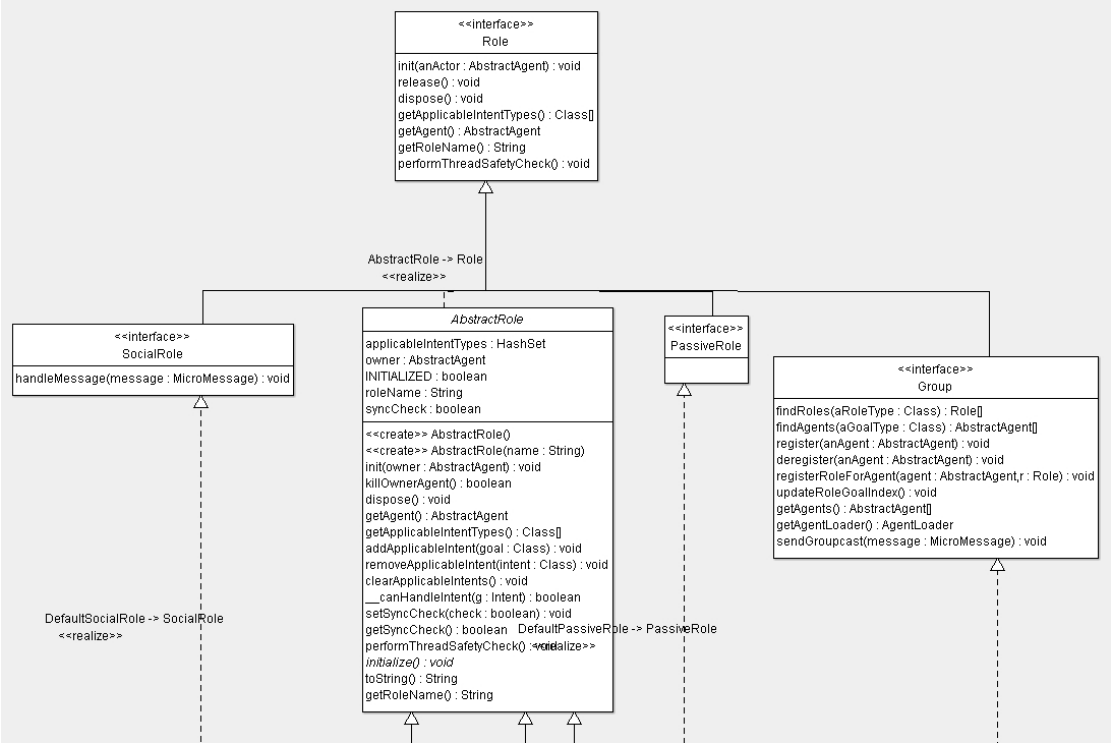


FIGURE A.6: Top part of Role hierarchy

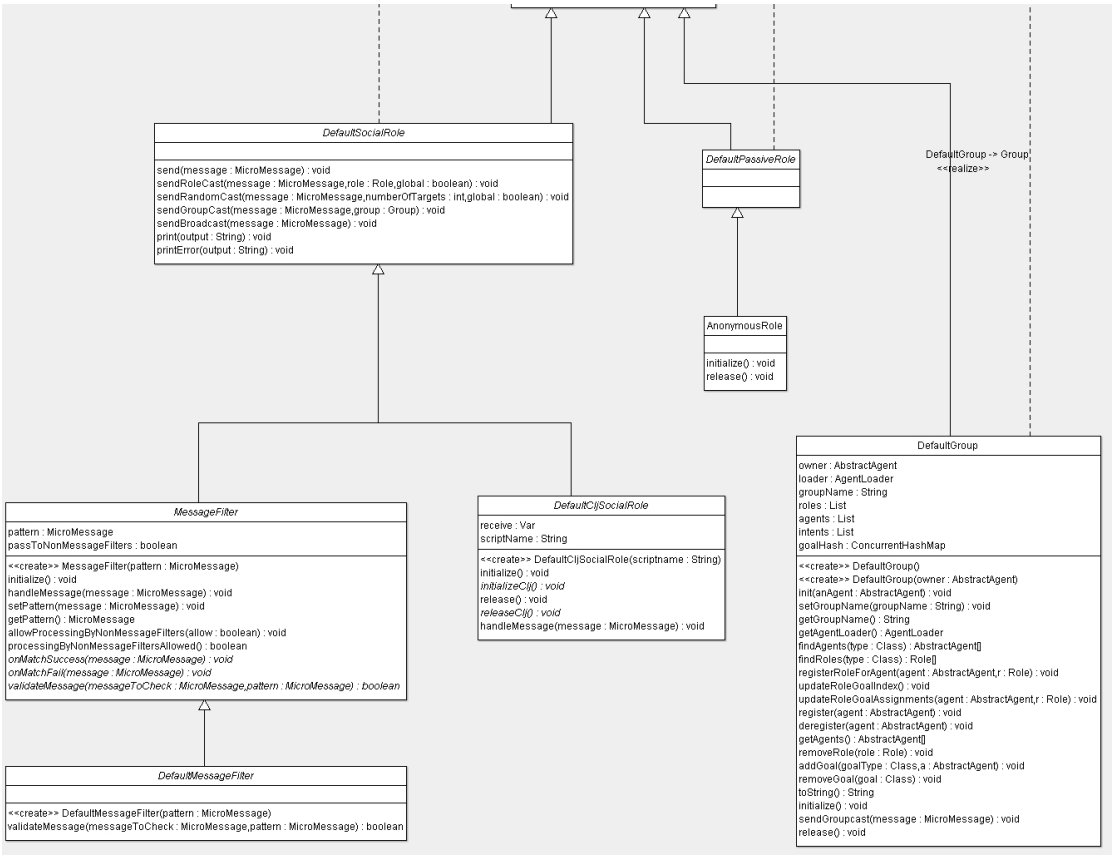


FIGURE A.7: Bottom part of Role hierarchy

Appendix B

Fairness Benchmark 'TalkingAnts'

B.1 Design

The application 'TalkingAnts' is based on an ants simulation created by Rich Hickey (see [Hic] for original implementation), developer of Clojure, to show the concurrency handling capabilities of Clojure's STM. Its design includes a two-dimensional grid as environment, an arbitrary number of ants which have a common nest, forage for food and return it to the nest – supported by indirect communication via pheromone. The environment keeps track of the food levels and simulates the evaporation of the pheromone (based on Clojure agents (see subsection 4.2.1 for brief introduction)).

In order to show the complementary use of Java and Clojure in μ^2 the logic of the ants has been removed and only the environmental elements (such as the two-dimensional grid, food and evaporation functionality) have been retained in a modified version.

The ant logic itself has been reimplemented as Java-based agent roles which access the Clojure environment directly (via the ClojureConnector provided as part of μ^2). In consequence the application builds on the developed platform and can be controlled via Clojure functions directly accessing the platform management functionality.

The basic ant logic is as follows: While the ant is in 'foraging mode' it proceeds

in directions which are weighted with a priority for food and pheromone (in this order). If neither exists they move in a random direction. If food is found ants switch into 'homing mode' and directions are weighted by home and pheromone.
As a further step the notion of direct communication (which constitutes the name of this application) is introduced to make use of the message passing mechanisms and to put more strain on scheduling fairness mechanisms. When activated ants can perceive other ants in a certain range. If two ants are in each other's range and the ants are in the 'opposite mode' (foraging vs. homing) the ants can exchange the coordinates of their last mode switch (i.e. where it had picked up food or dropped food (in the nest)). They follow the shortest path to those coordinates for a defined number of steps (e.g. 40 steps (to avoid blind commitment)) or until they have reached the coordinate (respectively satisfaction (e.g. picked up other food on the way)). It is needless to say that this functionality obviously breaks the principal ideas of ant simulations. However, apart from the use of both indirect as well as direct communication (as described in subsection 5.3.1) it serves another purpose:
Considering the alternative means of routing (shortest path for given destination vs. weighted/randomized routing) the processing-intensive direction-weighing procedure results in significantly longer processing times for ants which did not exchange coordinates. As such the extended version of this application produces largely differing processing times between different agents which helps to evaluate the effectiveness of different scheduling mechanisms. A screenshot of the application is shown in Figure B.1.
Ants are represented as (directed) lines. Red ants carry food, black ones do not (indicating their 'mode'). Red squares represent food (colour intensity indicates amount of food), the quadrat in the top left quadrant represents the (food-laden) nest. Pheromone is represented in green colour (again, intensity of colour indicates pheromone level), ants which have exchanged coordinates (and are actively heading towards those) are represented as blue squares.
Please bear in mind that this application is not tuned for efficiency in the shape of Ant Colony Optimization (ACO) algorithms.

As with all other code developed in the context of this work, instructions on its retrieval are provided in Appendix F.2.

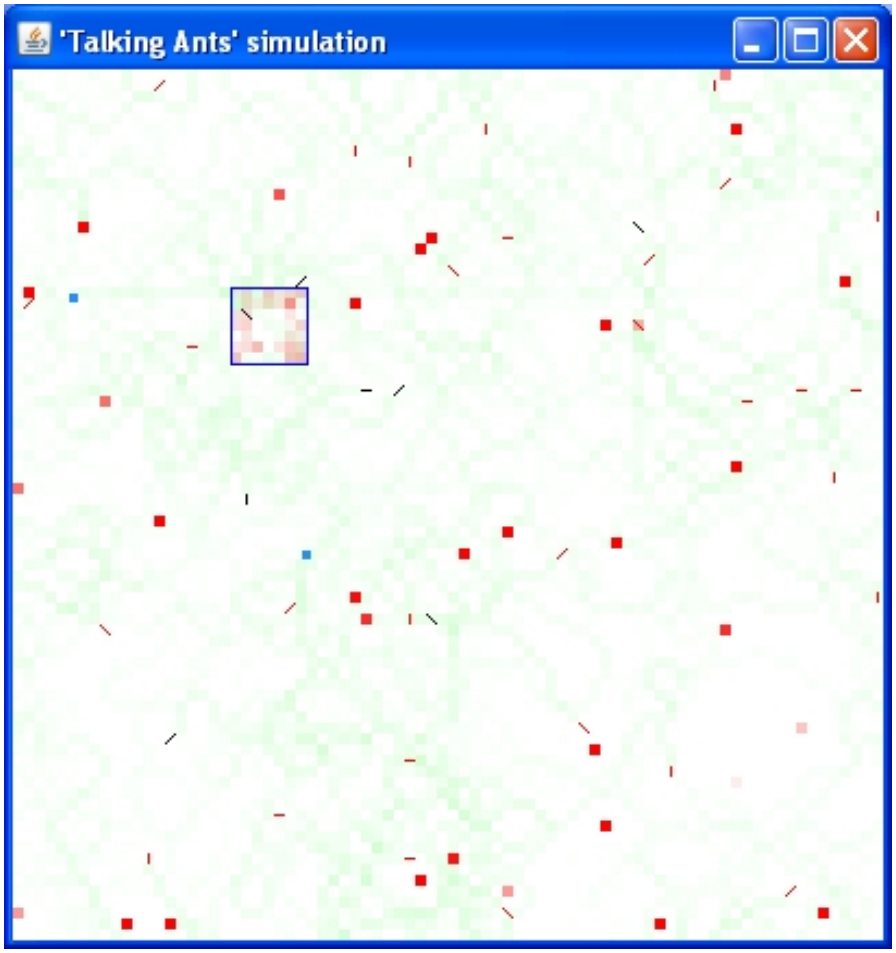


FIGURE B.1: Screenshot of 'TalkingAnts' simulation

B.2 Results

The 'TalkingAnts' application serves particularly well to compare performance as well as round-level fairness of schedulers. To measure this ants hold information on processed rounds and send this information to a central AntController which keeps track of the application runtime and calculates the fairness between different ants as standard deviation of rounds across all ants. Performance of the application can be measured as average rounds per minute (or other temporal unit). The AntController can be accessed from the Clojure REPL (i.e. command line).

For reliable measurement of the differences between scheduler implementations the talking capabilities have been switched off as the processing difference of the routing procedures (weight-based vs. directed) is amplified by the random situation

of ants meeting each other. The evaluation has been done for Java ThreadPools (which serve as a performance baseline), a modified version of the FairThreads framework (see description in subsection 5.3.2) and the multithreaded round robin scheduler developed in the context of this work. Table B.1 lists benchmark results for repeated runs (the values represent an average over five runs for each scheduler on the development machine as specified in Appendix F.1). Each run was undertaken in a clean JVM instance (i.e. shutdown of all Java applications).

Scheduling mechanism	**Java ThreadPool**	**FairThreads**	**Fair Task Scheduler (μ^2)**
Number of ants	50	50	50
Number of threads	50	4	4
Runtime: 10 min.			
Average number of rounds	430	148	295
Fairness	6.86	0.0	0.0
Rounds per minute	43	14.8	29.5
Runtime: 15 min.			
Average number of rounds	641	220	442
Fairness	7.6	0.0	0.0
Rounds per minute	42.7	14.7	29.5
Runtime: 20 min.			
Average number of rounds	850	290	584
Fairness	9.7	0.0	0.0
Rounds per minute	42.5	14.5	29.2

TABLE B.1: Performance and Fairness comparison of evaluated schedulers in 'Talking Ants' application

Appendix C

Multi-agent Platform Performance Benchmark

C.1 Design

The benchmark scenario used to evaluate the performance of selected multi-agent frameworks includes four agents, one of which (the 'user agent') requests a service from a service provider who himself coordinates two primitive agents to compose his functionality. The scenario in fact seeks to represent a simple printer service. Upon request data is collected and printed to the console before confirming the success of the execution. A benchmark round is initiated by the user agent requesting the service. Objective of this benchmark is to measure the interaction performance of the according frameworks. Thus the actual processing within the agents is very simplistic and involves no deliberation or other advanced features in order not to confound the results. The scenario is run for a parameterized number of rounds, its execution time is measured to indicate the framework performance. Figure C.1 depicts the interactions in the described scenario. Numbers indicate the execution order of one scenario round.

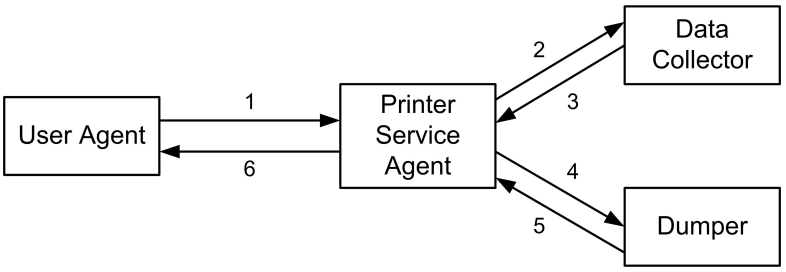


FIGURE C.1: Message flow in performance benchmark scenario

C.2 Results

This section lists some results of the benchmark runs. The benchmarks have been developed a total number of five platforms – including μ^2. The four other platforms are shortly characterized in the following. For a more in-depth discussion of those frameworks refer to [FNP10].

- *3APL* – An Abstract Agent Platform (3APL) [3APa] has been developed at the University of Utrecht and represents BDI-agents by implementing an extensive reasoning cycle which does not only allow simple goal activation upon events but additionally considers goal revision during execution (see [3APb] for details). It is the 'heaviest' agent architecture in this test.
- *JADE* – Java Agent Development Environment (JADE) [JAD09] is the only remaining popular full implementation of the FIPA specifications and is maintained by Telecom Italia since 1998. It implements own reasoning capabilities but also allows the implementation of so-called simple behaviour - which has been used for this test. It belongs to the group of platforms catering for the implementation of agents of strong notion.
- *OPAL* – OPAL represents the standard-compliant extension of μ^2. It encapsulates the micro-agent framework discussed throughout this work but has its own Message Transport mechanisms (mediated via JAS) which make it FIPA-compliant – similar to JADE.
- *MadKit* – MadKit represents an alternative approach to organizational MAS such as the micro-agent concept used here and has been developed at the University of Montpellier. In consequence MadKit is driven by a weak agent

understanding (" ... a class defining [a] basic life-cycle ..." [FG98]) and allows the resource-friendly use of agents.

The actual runs have been performed in logarithmic steps from 1000 to 1000000 rounds to evaluate the scalability of the different platforms. Interestingly, all platforms but MadKit and μ^2 show a significant performance drop with increasing number of benchmark rounds and either stop working (as the case for 3APL and JADE) or fall back to a slow processing performance (such as OPAL). MadKit and μ^2 show constant performance, clearly outlining the efficiency principle of small-scale agents respectively micro-agents. Contrasting the latter ones, μ^2 takes half the time of MadKit to run the benchmark. However, more than the actual performance the scalability behaviour is of concern for micro-agents, thus clustering those two platform with regards to the efficiency.

Table C.1 gives an outline of the performance results for all platforms mentioned. Dashes (–) indicate that the platform failed to perform the according number of rounds. Figure C.2 and C.3 visualize the performance over an increasing number of rounds. The latter figure is focusing on runs beyond 10000 rounds (and thus fades out the curve for the 3APL performance).

Each value listed here represents an average over ten runs for each configuration on the hardware as specified in Appendix F.1.

	JADE	3APL	MadKit	OPAL	Micro-agents
1000	0.25	1422	0.125	1.1	0.08
10000	8.3	–	1.025	10.4	0.41
100000	888.2	–	9.8	100.5	4.5
1000000	–	–	96	1035.7	42.2

Table C.1: Benchmark results for Agent Platforms per scenario rounds (in seconds)

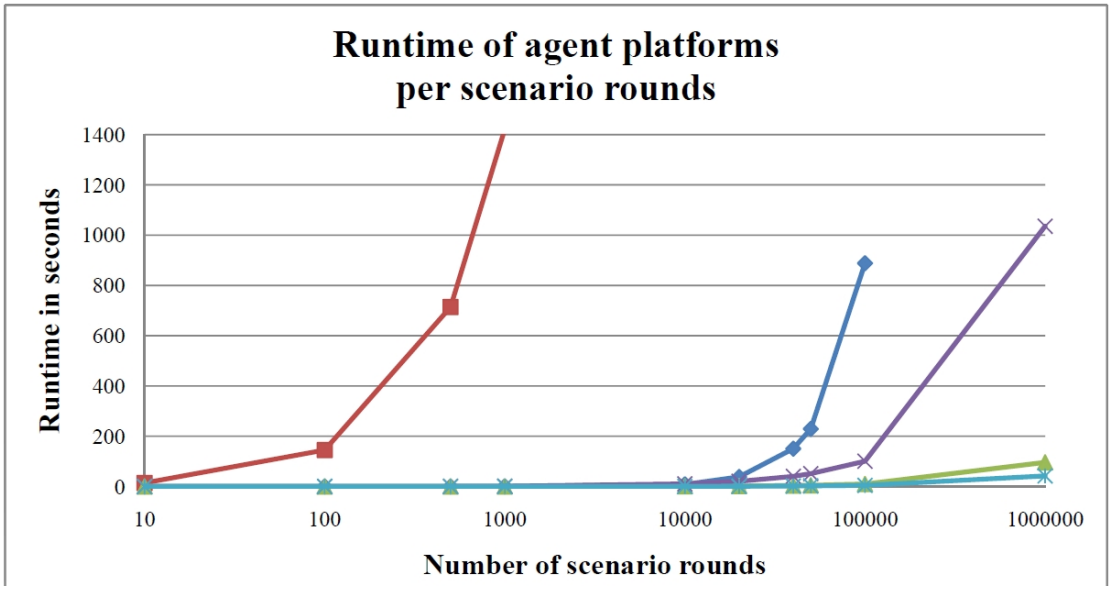


FIGURE C.2: Performance behaviour with increasing number of benchmark rounds

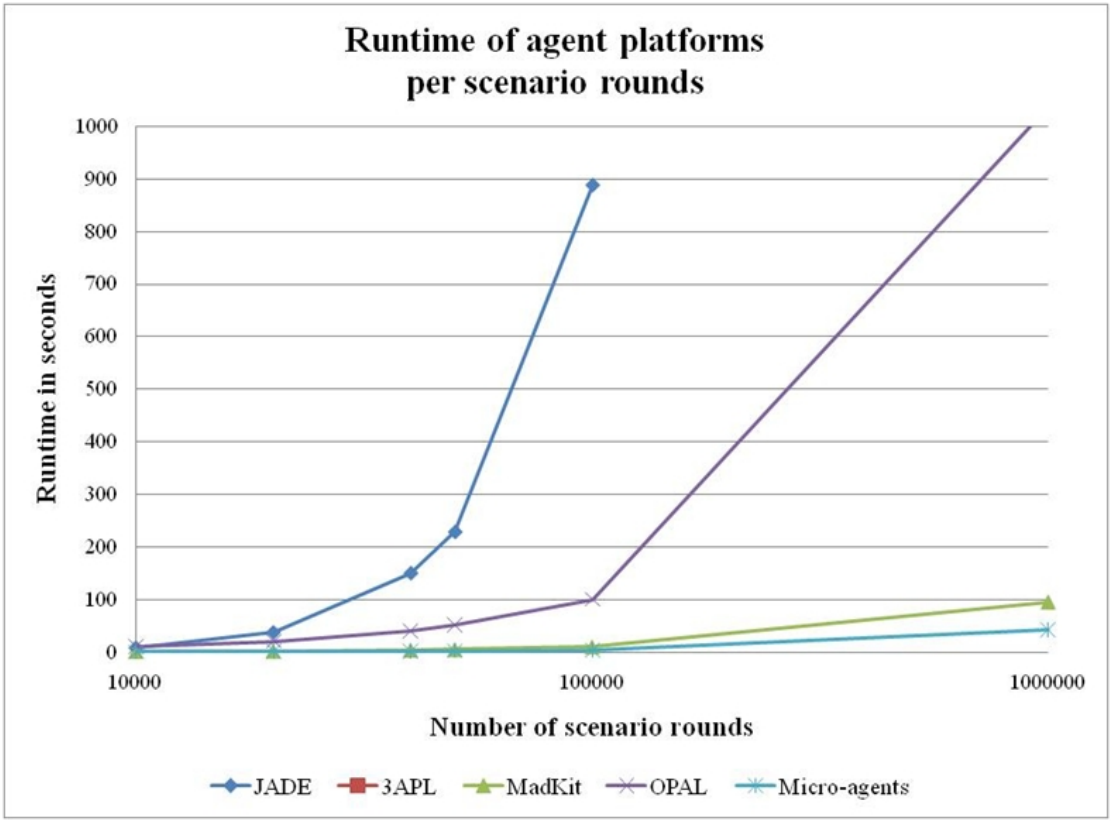


FIGURE C.3: Performance behaviour with increasing number of benchmark rounds (beyond 10000 rounds)

Appendix D

MOA Application Scenario and Performance Benchmark

D.1 MOA Application Scenario

To clarify the infrastructural similarities and application potential lying in the combined use of micro-agents with Android itself, a simple scenario will be described which could be supported by the current implementation of MOA.

In this scenario Android will be augmented with micro-agents to provide a simple response mechanism upon received Short Message Service (SMS) text messages and is visualized in Figure D.1. The numbers in the figure indicate the order of processing and are used for the following narrative description. The figure depicts the different application components on the Android side and the according counterparts in the micro-agent system. Center of MOA is the tight link between a dedicated micro-agent (role) – the AndroidInterfaceRole – and an according Android service – the MicroAgentInterfaceService. The latter subscribes to various system events, such as received SMS, while its micro-agent counterpart registers an applicable intent (AndroidIntent). The AndroidIntent (in the micro-agent package) resembles the structure of an actual Android intent, allowing the conversion to the latter by the AndroidInterfaceRole. Messages including this intent will always be resolved to this interfacing role, ensuring that the intent is eventually executed on Android. Given this brief understanding for the core architecture the scenario is to be described in more depth:

An incoming SMS (1) is received by the MicroAgentInterfaceService which has subscribed to this Android event. It is automatically converted to an ReceivedAndroidIntentEvent in the micro-agent platform and all micro-agents which have subscribed to it are notified (here: AndroidReceiverRole) (2). The AndroidReceiverRole itself makes use of intents to resolve an agent who is responsible for sending responses via SMS (3). The SMSResponder contacts a CalendarManager in order to check if the owner of the mobile phone is currently occupied (e.g. by appointment) and does so by accessing both the Android calendar (which makes an interaction with Android itself necessary (messages (5) to (8)) and optionally agents on a connected platform (e.g. desktop version of μ^2) to check for eventual appointments. Upon return to the SMSResponder (9), this finally replies to the sender of the original SMS again by making use of Android facilities (messages (10), (11)).

This simple scenario serves as an example where the micro-agent package can significantly enhance the functionality provided by Android and also show its potential to serve as a middleware between desktops and mobile devices. For micro-agents the interaction is seemless; both realms are hidden by the according interface roles/service but mutually allow full access to a wide range of functionality. Micro-agents are thus one solution to realize the vision of a practical use of agents on mobile platforms; not restricted to a simple portation of an agent platform to a mobile application platform but full exploitation of the underlying capabilities.

Additionally, performance advantages by using MOA are shown in the next section.

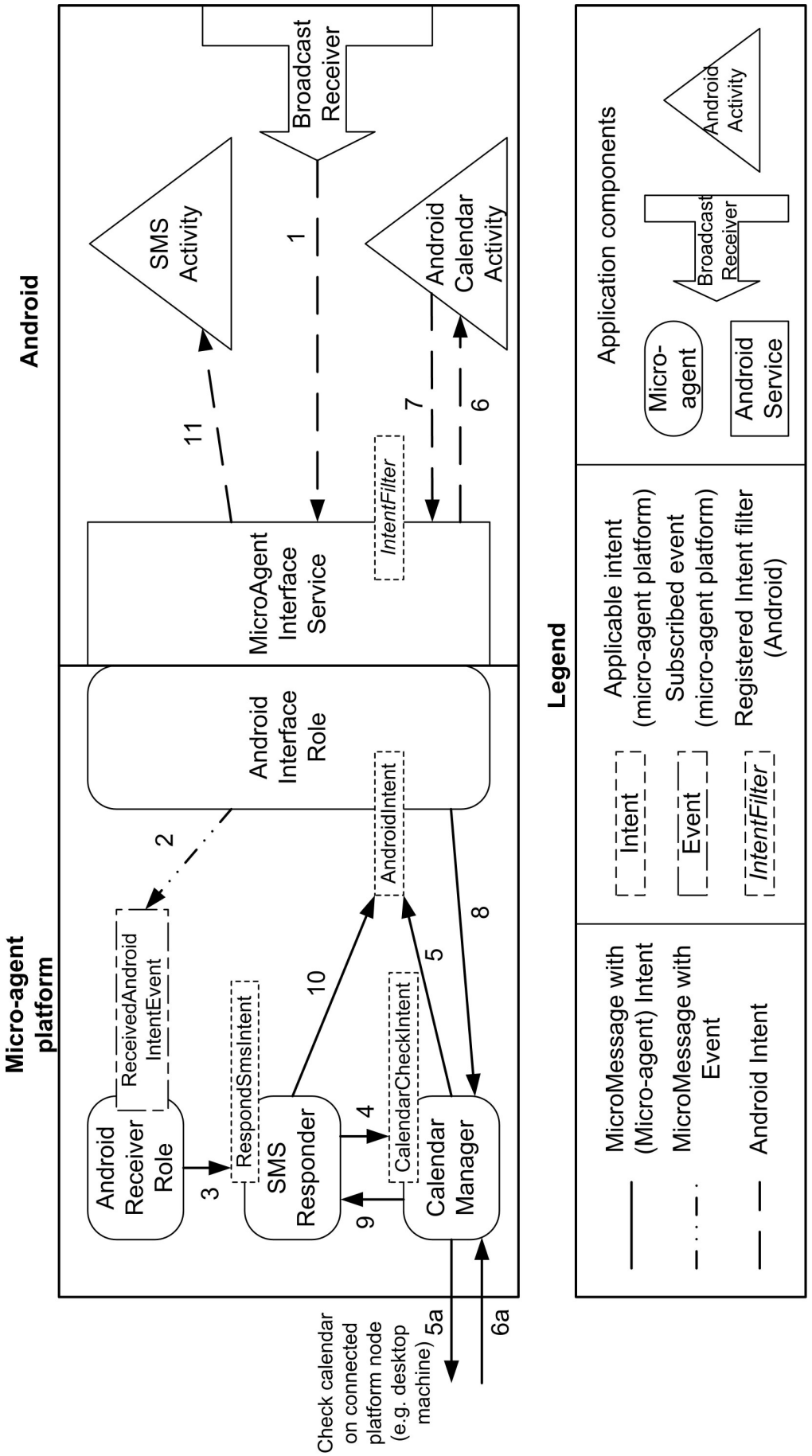


FIGURE D.1: Potential MOA application scenario

D.2 MOA Performance Benchmark

D.2.1 Design

To measure potential performance advantages of using MOA elements, respectively micro-agents, in contrast to conventional Android services a simple benchmark scenario has been produced which tries to isolate a pure Android-based approach from the blended use with MOA.

Similar to the earlier example application scenario this benchmark scenario (see Figure D.2[1]) emulates a automatic response application which receives a stimulus (e.g. SMS) which is then passed to a coordinating higher-level agent respectively service which then resolves the name for the message sender[2], e.g. via a list of contacts. The the priority of the sender is resolved (e.g. 'Is he/she suitable for automatic responses?' (e.g. important customer, annoying customer)) and then finally respond using a dedicated micro-agent (in MOA)/service (in Android) for this purpose. Looking at the number of messages, the conversion between Android intents and micro-agent messages enforces the use of two additional messages to complete one benchmark cycle. Both alternatives have the Benchmark Service as initializing component in common, all other functionality is delegated to the realm of the according technology. Similar to the agent platform benchmark (see Appendix C) the complexity of the agent/service internals is low and standardized to allow comparability and the isolation of the pure interaction performance of both technologies.

The performance is determined by measuring the execution time for a parameterized number of executed scenario rounds. Both scenarios are run on an Android Emulator as specified in Appendix F.1 and after an initial warm-up run of 20 rounds for both (in order to start up all services and agents).

D.2.2 Results

The results of the benchmark run indicate a consistently higher performance of the MOA version of the benchmark. Native Android takes consistently at least

[1]The numbers in the figure indicate order of execution for each of the two technologies.

[2]Practical reason: The Protocol Data Units (PDU) containing the actual SMS do not include other (useful) information than the sender's phone number and the actual message.

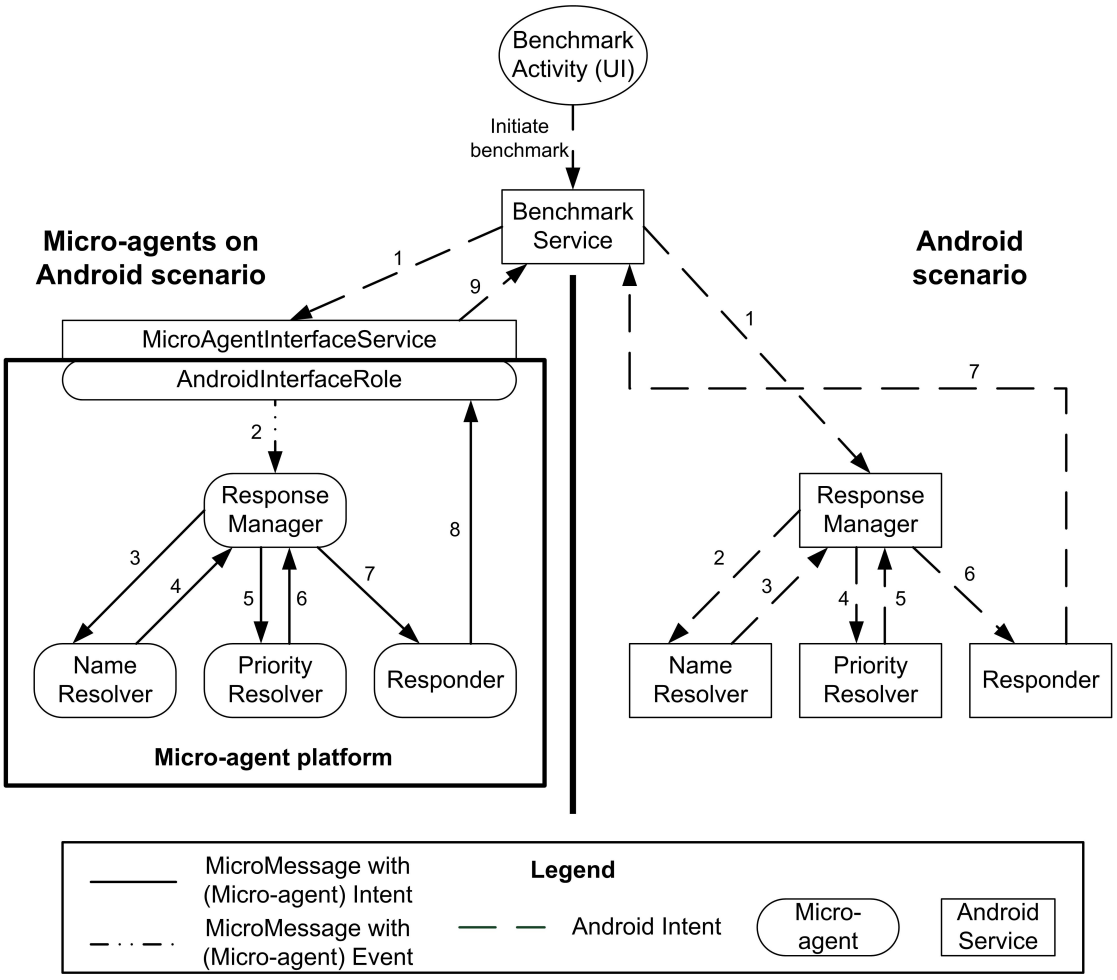


FIGURE D.2: MOA benchmark scenario

2.3 times as long as MOA to execute the scenario, as shown in Table D.1 and visualized in Figure D.3. As seen here, the combined use of MOA and Android does not only provide functionality advantages but also clear performance advantages. Apart from the functional extensions the use of MOA can simply be considered for performance reasons. However, the bottleneck would eventually be a frequent conversion between Android and micro-agents; applications in which every component is likely to interact directly with the user, MOA might not be the optimal solution which would demand for specific comparisons. However, the benchmark indicates a case in which the use of agent technology can provide tractable advantages.

Rounds	μ-agents	Android Services	Relative difference to MOA
5	257	614	2.39
25	881	2359	2.68
50	1834	4328	2.36
100	3711	8529	2.30
250	9356	21979	2.35
500	17757	42446	2.39
1000	35901	88562	2.47
2500	91792	242253	2.64
5000	156384	465606	2.98

TABLE D.1: Performance benchmark results relative performance factors of Android intents to micro-agents (in ms)

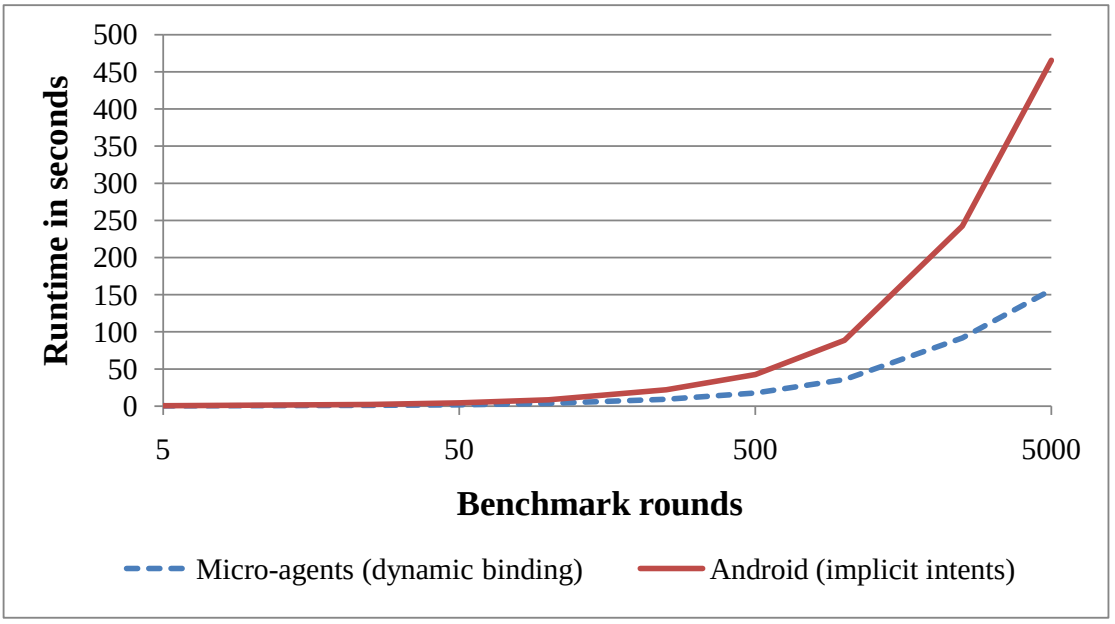


FIGURE D.3: Performance benchmark results

Appendix E

Data for 'Cultural Dimensions' Simulation Scenario

E.1 KNIME Analysis stream

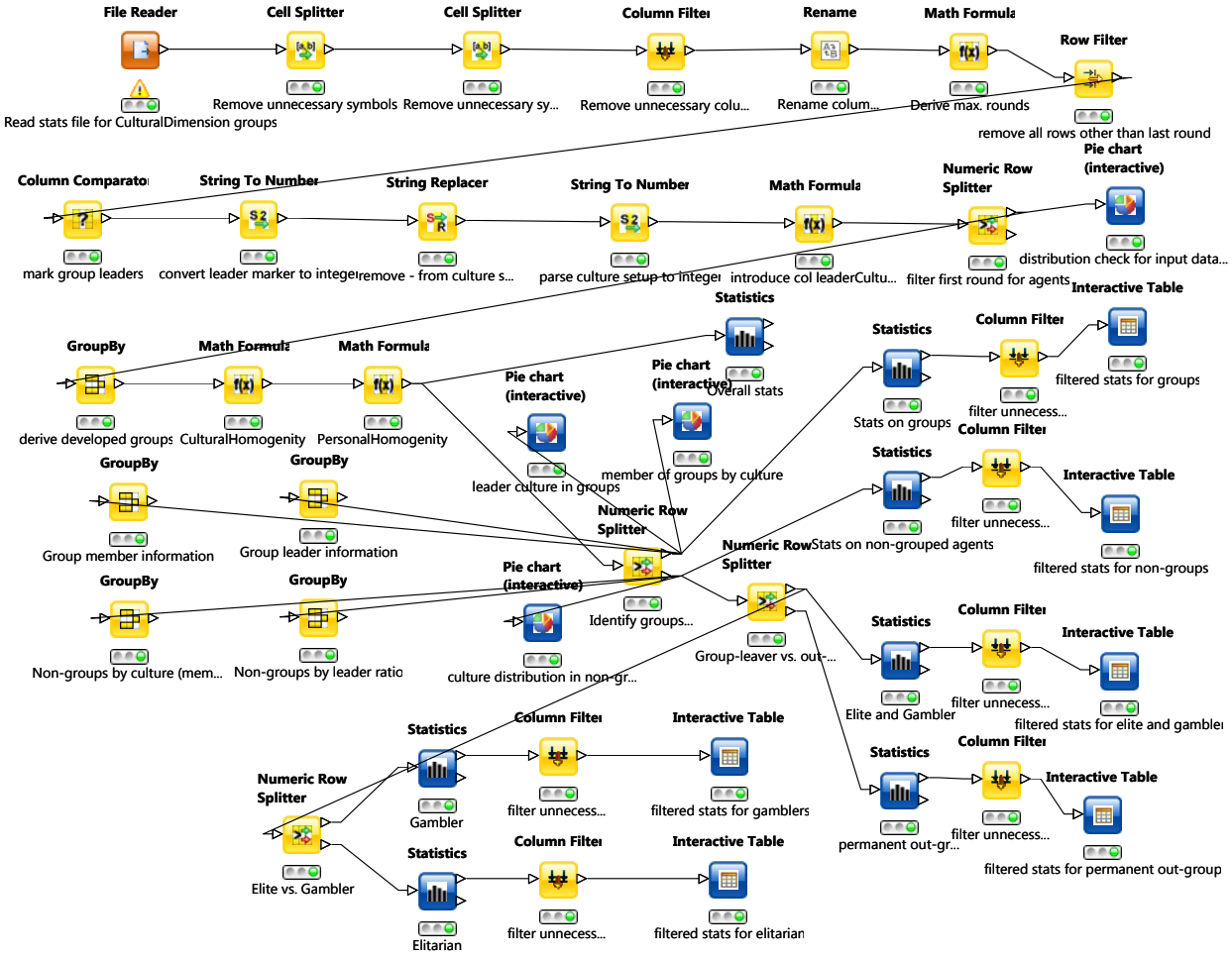


FIGURE E.1: KNIME Stream for analysis of simulation output

E.2 Results for Sensitivity Analysis

This section provides some results from the sensitivity analysis[1]. To show the differences outlined in the actual text both the results for an UAI weight factor of 1.5 as well 1.8 are provided at this point.

All tables are split into two parts. The basic structure includes the configured cultures, then overall statistics (e.g. number of out-group (ungrouped) agents, number of groups and the number of grouped agents). Below this general statistics on in-group agents are provided, along with the equivalent for ungrouped agents. The latter are then further divided into 'Elite' and 'Gambler' and 'Permanent out-group'. The former two cluster are both analysis as an aggregate as well as individually.

Common parameters for all tables in the following sections are listed in Table E.1.

Parameters	
Parameter	**Value**
UAI weight	1.5 or 1.8 (see according results)
PDI weight	1
IDV weight	1
Weight of rejections	0.1
Number of agents	200
Number of rounds	300
Number of group requests prior to application deliberation	3
Abbreviations	
Abbreviation	**Meaning**
UAI	Uncertainty Avoidance value
PDI	Power Distance value
IDV	Individualism value
RLC	number of rounds since last group change of agent
stddev	standard deviation
X	Placeholder for any cultural value (in cultural coordinates)

Table E.1: Global parameter settings and abbreviations

[1]The runs undertaken were significantly more extensive. However, the results presented here are representative for the model behaviour.

E.2.1 Results for UAI Weight Factor 1.5

Please see the results on the following page (as of lacking space on this one). For information on global parameters and abbreviations used in the result tables please refer to the beginning of this section.

Culture	neutral	uncertainty-accepting	uncertainty-avoiding	liberal	status-oriented	collectivist	individualist
mean UAI	0	**-4**	**4**	0	0	0	0
mean PDI	0	0	0	**-4**	**4**	0	0
mean IDV	0	0	0	0	0	**-4**	**4**
ungrouped agents	107	164	83	78	134	78	133
ungrouped percentage	**0.535**	**0.82**	**0.415**	**0.39**	**0.67**	**0.39**	**0.665**
mean group size	2.3	2.333	2.34	2.392	2.167	2.521	2.129
group size stddev	0.648	0.816	0.626	0.695	0.461	0.875	0.428
no of groups	40	15	50	51	30	48	31
grouped agents	**93**	**36**	**117**	**122**	**66**	**122**	**67**
mean in-group status	1.991	2.131	1.92	2.013	1.842	1.957	1.968
status stddev	0.569	0.498	0.527	0.461	0.492	0.56	0.591
mean RLC	287	251.1	294.28	290.155	286.4	289.921	287.4
minimal RLC	245.5	2	248	260.5	260	234.5	250
mean UAI	0.395	-1.998	3.498	0.262	0.3	0.018	0.315
mean PDI	-0.335	-0.956	-0.353	-4.138	2.814	-0.119	-0.866
mean IDV	-0.771	-1.262	-0.413	-0.368	-1.014	-4.419	3.056
mean out-group status	2.103	2.037	2.253	2.103	2.127	2.154	2.09
status stddev	0.8	0.798	0.763	0.8	0.789	0.757	0.83
mean RLC	117.2	30.14	261.3	114.987	96.276	220	118.91
minimal RLC	0	0	37	0	0	113	0
mean UAI	-0.738	-4.72	4.084	-1.051	-0.47	-0.679	-0.496
mean PDI	0.542	0.36	0.663	-3.538	4.746	0.59	0.579
mean IDV	0.551	0.262	0.747	0.59	0.366	-3.385	4.391
'Elite' and 'gamblers'	97	164	29	72	125	68	117
percentage of all	0.485	0.82	0.145	0.36	0.625	0.34	0.585
mean status	2.134	2.037	2.621	2.097	2.144	2.103	2.068
status stddev	0.812	0.798	0.561	0.8	0.8	0.775	0.828
mean RLC	98.4	30.14	188.579	99.569	81.6	208.338	94.145
minimal RLC	0	0	37	0	0	113	0

TABLE E.2: Sensitivity Analysis with UAI weight factor 1.5 (1/2)

mean UAI	-1.134	-4.72	2.828	-1.417	-0.664	-1.162	-1	
mean PDI	0.794	0.36	2.517	-3.444	4.952	0.882	0.735	
mean IDV	0.649	0.262	1.345	0.722	0.328	-3.338	4.368	
'Elite' individuals	30	10	24	19	32	29	37	
percentage of all	0.15	0.05	0.12	0.095	0.16	0.145	0.185	
mean status	2.333	2.6	2.583	2.211	2.594	2.034	2.189	
status stddev	0.711	0.516	0.584	0.787	0.56	0.778	0.845	
mean RLC	236.067	224.2	212.417	265	210	238.759	215.676	
minimal RLC	164	207	47	168	76	163	145	
mean UAI	2.733	0.8	3.375	3	3.4	0.862	2.622	
mean PDI	1.767	1.2	2.542	-2.053	5.375	1	1.324	
mean IDV	2.133	3	1.583	1	1.469	-2.69	5.297	
'Gambler' individuals	67	154	5	53	93	39	80	
percentage of all	0.335	0.77	0.025	0.265	0.465	0.195	0.4	
mean status	2.045	2	2.8	2.057	1.989	2.154	2.012	
status stddev	0.843	0.8	0.447	0.818	0.814	0.779	0.819	
mean RLC	36.791	17.539	75.2	40.264	37.419	185.7	37.938	
minimal RLC	0	0	37	0	0	113	0	
mean UAI	-2.866	-5.078	0.2	-3	-2.065	-2.667	-2.688	
mean PDI	0.358	0.305	2.4	-3.943	4.806	0.795	0.462	
mean IDV	-0.015	0.084	0.2	0.623	-0.065	-3.821	3.938	
'Permanent out-group' individuals	10	0	54	6	9	10	16	
percentage of all	0.05	0	0.27	0.03	0.045	0.05	0.08	
mean status	1.8		2.056	2.167	1.889	2.5	2.25	
status stddev	0.632		0.787	0.753	0.6	0.527	0.856	
mean RLC	300		300	300	300	300	300	
minimal RLC	300		300	300	300	300	300	
mean UAI	3.1		4.759	3.333	2.222	2.6	3.25	
mean PDI	-1.9		-0.333	-4.667	1.889	-1.4	-0.562	
mean IDV	-0.4		0.426	-1	0.889	-3.7	4.562	

TABLE E.3: Sensitivity Analysis with UAI weight factor 1.5 (2/2)

E.2.2 Results for UAI Weight Factor 1.8

Please see the results on the following page (as of lacking space on this one). For information on global parameters and abbreviations used in the result tables please refer to the beginning of this section.

Culture	neutral	uncertainty-accepting	uncertainty-avoiding	liberal	status-oriented	collectivist	individualist
mean UAI	0	**-4**	**4**	0	0	0	0
mean PDI	0	0	0	**-4**	**4**	0	0
mean IDV	0	0	0	0	0	**-4**	**4**
ungrouped agents	107	163	90	101	128	80	133
ungrouped percentage	**0.535**	**0.815**	**0.45**	**0.505**	**0.64**	**0.4**	**0.665**
mean group size	2.359	2.188	2.245	2.25	2.182	2.791	2.167
group size stddev	0.778	0.544	0.63	0.576	0.528	1.226	0.379
no of groups	39	16	49	44	33	43	30
grouped agents	**93**	**37**	**110**	**99**	**72**	**120**	**67**
mean in-group status	1.942	2.151	1.974	2.093	1.917	1.929	2.072
status stddev	0.551	0.595	0.578	0.512	0.574	0.489	0.563
mean RLC	288.49	273.7	293.7	290	286	291.984	277.4
minimal RLC	264.5	200	236.5	262	244	264.5	199
mean UAI	0.53	-1.594	3.738	0.456	0.341	-0.056	0.706
mean PDI	-0.7	-1.047	-0.176	-4.036	2.735	-0.412	-0.906
mean IDV	-0.349	-1.469	-0.718	-0.527	-1.152	-4.453	3.25
mean out-group status	2.14	2.025	2.167	2	2.109	2.225	2.03
status stddev	0.782	0.793	0.768	0.787	0.796	0.729	0.797
mean RLC	104	33.27	276.4	239	87.375	191.775	105
minimal RLC	0	0	103	151	0	2	0
mean UAI	-0.832	-4.742	3.9	-0.851	-0.539	-0.625	-0.639
mean PDI	0.766	0.294	0.478	-3.792	4.891	0.962	0.534
mean IDV	0.393	0.288	0.756	0.347	0.578	-3.212	4.301
'Elite' and 'gamblers'	97	160	26	68	118	70	117
percentage of all	0.485	0.8	0.13	0.34	0.59	0.35	0.585
mean status	2.155	2.019	2.577	1.956	2.144	2.243	2.043
status stddev	0.782	0.789	0.578	0.781	0.787	0.731	0.803
mean RLC	83.9	28.26	218.3	209.5	69.356	176.3	78.4
minimal RLC	0	0	103	151	0	2	0

TABLE E.4: Sensitivity Analysis with UAI weight factor 1.8 (1/2)

mean UAI	-1.268	-4.831	2.231	-2.221	-0.788	-1.2	-1.077
mean PDI	0.959	0.306	2.731	-3.6	5.042	1.314	0.821
mean IDV	0.412	0.225	1.231	0.5	0.551	-3.3	4.333
'Elite' individuals	30	9	21	14	30	25	35
percentage of all	0.15	0.045	0.105	0.07	0.15	0.125	0.175
mean status	2.3	2.556	2.667	1.786	2.567	2.2	2.229
status stddev	0.794	0.527	0.577	0.8	0.568	0.816	0.8
mean RLC	240.7	239.667	236.4	260.5	194	244.48	222.7
minimal RLC	170	220	165	177	116	2	137
mean UAI	2.767	0.889	2.952	1.286	3.267	1.64	2.714
mean PDI	1.867	0.778	2.762	-2	5.633	1.2	1.6
mean IDV	1.433	2.778	1.095	1.571	1.833	-3.48	5.571
'Gambler' individuals	67	151	5	54	88	45	82
percentage of all	0.335	0.755	0.025	0.27	0.44	0.225	0.41
mean status	2.09	1.987	2.2	2	2	2.267	1.963
status stddev	0.773	0.792	0.447	0.777	0.8	0.688	0.793
mean RLC	13.7	15.669	142.2	196	26.898	138	16.793
minimal RLC	0	0	103	151	0	3	0
mean UAI	-3.075	-5.172	-0.8	-3.13	-2.17	-2.778	-2.695
mean PDI	0.552	0.278	2.6	-4.019	4.841	1.378	0.488
mean IDV	-0.045	0.073	1.8	0.222	0.114	-3.2	3.805
'Permanent out-group' individuals	10	3	64	33	10	10	16
percentage of all	0.05	0.015	0.32	0.165	0.05	0.05	0.08
mean status	2	2.333	2	2.091	1.7	2.1	1.938
status stddev	0.816	1.155	0.777	0.8	0.823	0.738	0.772
mean RLC	300	300	300	300	300	300	300
minimal RLC	300	300	300	300	300	300	300
mean UAI	3.4	0	4.578	1.97	2.4	3.4	2.562
mean PDI	-1.1	-0.333	-0.438	-4.182	3.1	-1.5	-1.562
mean IDV	0.2	3.667	0.562	0.03	0.9	-2.6	4.062

TABLE E.5: Sensitivity Analysis with UAI weight factor 1.8 (2/2)

E.3 Results for Multi-Cultural Setup

Please see the results on the following page (as of lacking space on this one). For information on global parameters and abbreviations used in the result tables please refer to Appendix E.2.

Culture	0;0;0	4;-4;4	-4;4;-4	4;-4;-4	4;4;-4	-4;4;4	-4;-4;4	-4;-4;-4	4;4;4
absolute member fraction	0.115625	0.125	0.0875	0.196875	0.14375	0.034375	0.075	0.140625	0.08125
absolute leader fraction	0.15748	0.070866	0.141732	0.133858	0.023622	0.07874	0.173228	0.188976	0.023622
leader/member ratio	0.540541	0.225	0.642857	0.269841	0.065217	0.909091	0.916667	0.533333	0.115385
average group size by leader	2.45	2.333	2.778	2.647	2	3	2.136	2.583	2.333
out-group individuals	35	32	44	9	26	61	48	27	46
grouped members	37	40	28	63	46	11	24	45	26
in-group fraction	0.513889	0.555556	0.388889	0.875	0.638889	0.152778	0.333333	0.625	0.361111
group size	2.45	2.333	2.091	2.647	2	3	2.136	2.583	2.333
group leaders	20	9	18	17	3	10	22	24	3
group leader fraction	0.277778	0.125	0.25	0.236111	0.041667	0.138889	0.305556	0.333333	0.041667

TABLE E.6: Group member and leader distribution for all interacting cultures

Culture coordinate	-4;X;X	4;X;X	X;-4;X	X;4;X	X;X;-4	X;X;4
Group member fraction (absolute)	0.3375	0.546875	0.5375	0.346875	0.56875	0.315625
Group leader fraction (absolute)	0.582677	0.251969	0.566929	0.267717	0.488189	0.346457
Group member fraction (relative)	0.140846	0.40908	0.232776	0.254011	0.291093	0.206275
Group leader fraction (relative)	0.196654	0.137795	0.304724	0.092864	0.277657	0.10935
out-group fraction	0.6625	0.453125	0.4625	0.653125	0.43125	0.684375
in-group fraction	0.375	0.607639	0.597222	0.385417	0.631944	0.350694
out-group fraction	0.625	0.392361	0.402778	0.614583	0.368056	0.649306
group size	2.4525	2.32825	2.42475	2.356	2.33025	2.4505

TABLE E.7: Aggregated group member and leader properties by cultural coordinate

Appendix F

Development Environment & Source Code Information

F.1 Development Environment Specifications

The hardware and software environment used for the development and all benchmark tests is listed as following:

Hardware	Intel Core2Duo (4 cores) at 2.66 Ghz, 3.25 GB RAM
Operating System	Microsoft Windows XP SP3
Java Development Kit [Java]	1.6.0 build 22

Network tests were undertaken using a virtual machine using Oracle VirtualBox and running an Ubuntu Linux. When measurements included console output (such as the multi-agent platform benchmarks (see Appendix C)), output was done on the host machine as of the high additional performance penalties for such operations on virtual machines.

The version information for this and further software used is listed at this point:

- Oracle VirtualBox 3.2.6 (with Ubuntu Linux 9.1) [Vir]
- Clojure 1.2 [Hic10a]
- Jetlang 0.2.1 [Jet]
- Netty 3.2.3 [Leea]

- XStream 1.3.1 [XSt]
- Eclipse 3.5.2 [Ecl]
- Android SDK with Emulator for Android 2.2 [Anda]
- KNIME 2.2.2 [KNI]

F.2 Information on Platform and Simulation Code

The sources for the platform code and developed applications are largely provided for public download.

- The platform code for μ^2 and MOA is provided (and will be maintained) under http://www.micro-agents.net.
- The CulturalDimensions simulation code as well as the 'TalkingAnts' application is provided (along with setup description) under http://culture.micro-agents.net.
- The code for the Asynchronous Message Passing framework benchmark is maintained at http://www.nzdis.org under the 'Opal' project, respectively available upon request to the author.
- The code for the Multi-agent Platform benchmark is available upon request to the author.

Bibliography

[3APa] 3APL Homepage. `http://www.cs.uu.nl/3apl/`. Accessed on: 15th September 2010.

[3APb] The cyclic interpreter (deliberation cycle) for 3APL agents. `www.cs.uu.nl/3apl/deliberationcycle.pdf`. Accessed on: 15th September 2010.

[ACBR06] L. Autunes, H. Coelho, J. Balsa, and A. Respicio. e*plore v.0: Principia for Strategic Exploration of Social Simulation Experiments Design Space. *Advancing Social Simulation: The First World Congress*, pages 295–306, 2006.

[Acta] ActorFoundry. `http://osl.cs.uiuc.edu/af/`. Accessed on: 25th July 2010.

[Actb] Actors Guild Framework. `http://actorsguildframework.org/`. Accessed on: 25th July 2010.

[Ada07] C. Adam. *Emotions: From psychological theories to logical formalization and implementation in a BDI agent.* PhD thesis, INP Toulouse, France, 2007.

[AG99] Ken Arnold and James Gosling. *The Java Programming Language, Second Edition.* Addison-Wesley, 1999.

[Agh86] G.A. Agha. *ACTORS: A Model of Concurrent Computation in Distributed Systems.* MIT Press, 1986.

[Anda] Android Emulator. `http://developer.android.com/guide/developing/tools/emulator.html`. Accessed on: 15th September 2010.

[Andb] Application Fundamentals. `http://developer.android.com/guide/topics/fundamentals.html`. Accessed on: 15th September 2010.

[Andc] Intents and Intent Filters. `http://developer.android.com/guide/topics/intents/intents-filters.html`. Accessed on: 15th September 2010.

[Andd] What is Android? `http://developer.android.com/guide/basics/what-is-android.html`. Accessed on: 15th September 2010.

[Apr] Network Agents. `http://sourceforge.net/projects/networkagent/`. Accessed on: 15th September 2010.

[ARCJ09] Jorge Agero, Miguel Rebollo, Carlos Carrascosa, and Vicente Julin. Does android dream with intelligent agents? In Juan Corchado, Sara Rodrguez, James Llinas, and Jos Molina, editors, *International Symposium on Distributed Computing and Artificial Intelligence 2008 (DCAI 2008)*, volume 50 of *Advances in Soft Computing*, pages 194–204. Springer Berlin / Heidelberg, 2009.

[ATKS07] A. Adl-Tabatabai, C. Kozyrakis, and B. Saha. Unlocking Concurrency. *Queue*, 4(10):24–33, 2007.

[Aus62] John Langshaw Austin. *How to Do Things with Words*. Harvard University Press, 1962.

[AVWW96] J. Armstrong, R. Virding, C. Wikstrom, and M. Williams. *Concurrent Programming in Erlang*. Prentice Hall, Englewood Cliffs, 2nd edition edition, 1996.

[Axe97] R. Axelrod. *Simulating Social Phenomena*, chapter Advancing the art of simulation in the social sciences, pages 21–40. Springer, Berlin, 1997.

[BA06] M. Ben-Ari. *Principles of Concurrent and Distributed Programming.* Prentice-Hall, 2006.

[Bal10] P. Ball. The Earth Simulator. *New Scientist*, pages 48–51, 30th October 2010.

[BBCM99] C. Bäumer, M. Breugst, S. Choy, and T. Magedanz. Grasshopper — A universal agent platform based on OMG MASIF and FIPA standards. In Ahmed Karmouch and Roger Impley, editors, *First International Workshop on Mobile Agents for Telecommunication Applications (MATA'99)*, pages 1–18, Ottawa, Canada, October 1999. World Scientific Publishing Ltd.

[BCG07] F. Bellifemine, G. Caire, and D. Greenwood. *Developing Multi-Agent Systems with JADE.* John Wiley & Sons, 2007.

[Ber] J. Bertolucci. Google: Desktop PCs will be dead by 2013. `http://www.pcadvisor.co.uk/news/index.cfm?newsId=3214232`. Accessed on: 15th September 2010.

[BG88] A. Bond and L. Gasser. *Readings in Distributed Artificial Intelligence.* Morgan Kaufmann, Los Angeles, CA, 1988.

[BH] R.H. Bordini and J.F. Hübner. Jason Manual. `http://jason.sourceforge.net/Jason.pdf`. Accessed on: 15th September 2010.

[BH09] R.H. Bordini and J.F. Hübner. *Multi-Agent Systems: Simulation and Applications*, chapter Agent-Based Simulation Using BDI Programming in Jason, pages 451–476. CRC Press, 2009.

[BHG06] Steve S. Benfield, Jim Hendrickson, and Daniel Galanti. Making a strong business case for multiagent technology. In *AAMAS*, pages 10–15, 2006.

[BHW07] R.H. Bordini, J.F. Hübner, and M. Wooldridge. *Programming Multi-Agent Systems in AgentSpeak using Jason*. Wiley, 2007.

[Boo94] G. Booch. *Object-oriented Analysis and Design with Applications*. Addison-Wesley, 1994.

[Bou] Frederic Boussinot. Fair Threads. `http://www-sop.inria.fr/meije/rp/FairThreads/`. Accessed on: 15th September 2010.

[BPL+06] L. Braubach, A. Pokahr, W. Lamersdorf, K.-H. Krempels, and P.-O. Woelk. A generic time management service for distributed multi-agent systems. *Applied Artificial Intelligence*, 20(2):229–249, 2006.

[Bra87] Michael E. Bratman. *Intention, Plans and Practical Reason*. CSLI Publications, Stanford University, 1987.

[Bro86] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2:14–23, 1986.

[Bro90] Rodney A. Brooks. Elephants Don't Play Chess. *Robotics and Autonomous Systems*, 6:3–15, 1990.

[Bru] Mike Brunt. Sun JVM RAM Utilization Limit On Windows Explained. `http://www.webapper.com/blog/index.php/2005/12/27/20051227104717/`. Accessed on: 15th September 2010.

[Bun77] M. Bunge. The GST challenge to the classical philosophies of science. *International Journal on General Systems*, 4:29–37, 1977.

[Bun79] M. Bunge. *Treatise on Basic Philosophy - Volume IV - Ontology II: A World of Systems*. D. Reidel Publishing Company, 1979.

[B.V] Tiobe Software B.V. TIOBE Programming Community Index. `http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html`. Accessed on: 15th September 2010.

[BWH07] Rafael H. Bordini, Michael Wooldridge, and Jomi Fred Hübner. *Programming Multi-Agent Systems in AgentSpeak using Jason (Wiley Series in Agent Technology)*. John Wiley & Sons, 2007.

[Can] Canalys. Android smart phone shipments grow 886% year-on-year in Q2 2010. `http://www.canalys.com/pr/2010/r2010081.html`. Accessed on: 15th September 2010.

[Cas95] C. Castelfranchi. Guarantees for autonomy in cognitive agent architecture. In *Proceedings of the Workshop on Agent Theories, Architectures and Languages, ATAL '94*, volume 890 of *LNAI*, pages 56–70. Springer, 1995.

[CBF04] C. Carabelea, O. Boissier, and A. Florea. Autonomy in multi-agent systems: A classfication attempt. In M. Nickles, M. Rovatsos, and G. Wei, editors, *Agents and computational autonomy: Potential, Risks, and Solutions*, volume 2969 of *LNCS*, pages 103–113. Springer, 2004.

[Clo] Java Interop. `http://clojure.org/java_interop`. Accessed on: 15th October 2010.

[Cof] N. Coffey. Randomness of bits with LCGs. `http://www.javamex.com/tutorials/random_numbers/lcg_bit_positions.shtml`. Accessed on: 15th September 2010.

[Coh91] P.R. Cohen. A Survey of the Eighth National Conference on Artificial Intelligence: Pulling together or pulling apart? *AI Magazine*, 12(1):16–41, 1991.

[Col90] J.S. Coleman. *The foundations of Social Theory*. Harvard University Press, Boston, 1990.

[Con05] J. Conklin. *Dialogue Mapping: Building Shared Understanding of Wicked Problems*. Wiley, 2005.

[Cre93] D. Crevier. *AI. The Tumultuous History of the Search for Artificial Intelligence*. Basic Books, New York, 1993.

[Das08] M. Dastani. 2apl: a practical agent programming language. *Autonomous Agents and Multi-Agent Systems*, 16:214–248, 2008. 10.1007/s10458-008-9036-y.

[Den87] D.C. Dennett. *The Intentional Stance*. MIT Press, Cambridge, Massachusetts, 1987.

[DG94] J.E. Doran and N. Gilbert. *Simulating Societies: The Computer Simulation of Social Phenomena*, chapter Simulating societies: An introduction, pages 1–18. UCL Press, London, 1994.

[Dij65] E. W. Dijkstra. Solution of a Problem in Concurrent Programming Control. *Communications of the Association of Computing Machinery*, 8(9):569, 1965.

[DVD08] C. Deissenberg, S. Vanderhoog, and H. Dawid. EURACE: A massively parallel agent-based model of the European economy. *Applied Mathematics and Computation*, 204(2):541–552, October 2008.

[DVM02] Alexis Drogoul, Diane Vanbergue, and Thomas Meurisse. Multi-agent based simulation: Where are the agents? In *Multi-Agent-Based Simulation II*, Berlin, Heidelberg, New York, 2002. Third International Workshop, MABS 2002 Bologna, Italy, July 2002, Springer.

[EA96] J.M Epstein and R. Axtell. *Growing Artificial Societies - Social Science from the Bottom Up*. MIT Press, 1996.

[Ecl] Eclipse.org. `http://www.eclipse.org/`. Accessed on: 15th September 2010.

[Eco] The Economist. Agents of Change. `http://www.economist.com/node/16636121`. Accessed on: 15th September 2010.

[EEHT07] Bruce Edmonds, Bruce Edmonds, Cesareo Hernandez, and Klaus G. Troitzsch. *Social Simulation: Technologies, Advances and New Discoveries*. IGI Global, 2007.

[EM88] R.S. Englemore and T. Morgan, editors. *Blackboard Systems*. Addison-Wesley, 1988.

[EW95] O. Etzioni and D. Weld. Intelligent agents on the Internet: Fact, Fiction and Forecast. *IEEE Expert*, 10:44–49, 1995.

[Fau97] S.R. Faulk. Software requirements: A tutorial. In R. Thayer and M. Dorfman, editors, *Software Requirements Engineering*, chapter Software Requirements: A Tutorial. IEEE Computer Press, 1997.

[Fer99] Jacques Ferber. *Multi-Agent Systems - An Introduction to Distributed Artificial Intelligence*. Addison-Wesley, 1999.

[FFMM94] Tim Finin, Richard Fritzson, Don McKay, and Robin McEntire. KQML as an agent communication language. In *CIKM '94: Proceedings of the third international conference on Information and knowledge management*, pages 456–463, New York, NY, USA, 1994. ACM.

[FG96] S. Franklin and A. Graesser. Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agent. *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Language*, 1996.

[FG98] J. Ferber and O. Gutknecht. A meta-model for the analysis and design of organizations in multi-agent systems. *Third International Conference on Multi-Agent Systems (ICMAS '98), IEEE Computer Society*, pages 128–135, 1998.

[FIPa] FIPA - Wikipedia. `http://en.wikipedia.org/wiki/FIPA`. Accessed on: 15th September 2010.

[FIPb] FIPA ACL Message Structure Specification. `http://www.fipa.org/specs/fipa00061/SC00061G.html`. Accessed on: 15th September 2010.

[FIPc] FIPA Agent Message Transport Service Specification. `http://www.fipa.org/specs/fipa00067/SC00067F.html`. Accessed on: 15th September 2010.

[FIPd] FIPA-OS Agent Toolkit. `http://sourceforge.net/projects/fipa-os/`. Accessed on: 15th September 2010.

[FIPe] FIPA SL Content Language Specification. `http://www.fipa.org/specs/fipa00008/SC00008I.html`. Accessed on: 15th September 2010.

[FIPf] Foundation for Physical Intelligent Agents. `http://www.fipa.org`. Accessed on: 15th September 2010.

[FIP02] FIPA. *FIPA Communicative Act Library Specification*. Foundation for Intelligent Physical Agents (FIPA), 2002.

[FIP03] Publicly Available Implementations of FIPA Specifications. `http://www.fipa.org/resources/livesystems.html`, 2003. Accessed on: 15th September 2010.

[FIP04] FIPA. FIPA Agent Management Specification. `http://www.fipa.org/specs/fipa00023/SC00023K.html`, 2004. Accessed on: 10th October 2009.

[Fis95] P. Fishwick. *Simulation Model Design and Execution*. Prentice Hall, 1995.

[Fis96] P. Fishwick. Computer simulation: Growth through extension. *IEEE Potential*, February/March:24–27, 1996.

[FNP10] C. Frantz, M. Nowostawski, and M. Purvis. Multi-agent platforms and Asynchronous Message Passing. Information Science Discussion

Paper 2010/07, Dept. of Information Science, University of Otago, New Zealand, Dunedin, November 2010. `http://eprints.otago.ac.nz/1011/`.

[For71] J.W. Forrester. *World Dynamics*. MIT Press, 1971.

[FPV98] A. Fuggetta, G. Picco, and G. Vigna. Understanding Code Mobility. *IEEE Transactions on Software Engineering*, 24(5):342–361, 1998.

[Fra09] C. Frantz. Unifying micro-agent communication in the Otago Agent Platform (OPAL), 2009. `http://eprints.otago.ac.nz/874/`.

[GC10] N. Griffiths and K. Chao, editors. *Agent-Based Service-Oriented Computing*. Springer, 2010.

[Gen] M. Genesereth. Knowledge Interchange Format. `http://logic.stanford.edu/kif/specification.html`. Accessed on: 15th September 2010.

[Geo09] M. Georgeff. The gap between software engineering and multi-agent systems: Bridging the Divide. *International Journal for Agent-Oriented Software Engineering*, 3(4):391–396, 2009.

[GFM00] O. Gutknecht, J. Ferber, and F. Michel. The MadKit Agent Platform Architecture. Technical report, Laboratoire d'Informatique, de Robotique et de Microelectronique de Montpellier, Universite Montpellier II, 2000.

[GHB00] M. Greaves, H. Holmback, and J. Bradshaw. *Issues in Agent Communication*, chapter What is a Conversation Policy?, pages 118–131. Springer, 2000.

[Gil95] N. Gilbert. *Artificial Societies: The Computer Simulation of Social Life*, chapter Emergence in social simulation, pages 144–156. UCL Press, London, 1995.

[Gil96] N. Gilbert. *Social Science Microsimulation*, chapter Simulation as a research strategy, pages 448–454. Springer, 1996.

[GK94] M.R. Genesereth and S.P. Ketchpel. Software Agents. *Communications of the ACM*, 37:48ff., 1994.

[GL87] M.P. Georgeff and A. L. Lansky. Reactive reasoning and planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, pages 677–682, Seattle, WA, USA, 1987.

[Goo10] Google I/O 2010 - A JIT Compiler for Android's Dalvik VM. `http://www.youtube.com/watch?v=Ls0tM-c4Vfo`, 2010. Accessed on: 08th October 2010.

[GT05] N. Gilbert and K.G. Troitzsch. *Simulation for the Social Scientist*. Open University Press, 2005.

[Gur63] G. Gurvitch. *La vocation actuelle de la sociologie*. PUF, 1963.

[Gur64] G. Gurvitch. *Spectrum of social time*. Springer, 1964.

[Has] Haskell. `http://haskell.org/`. Accessed on: 15th September 2010.

[HB88] G. Hofstede and M.H. Bond. The Confucius Connection: From Cultural Roots to Economics Growth. *Organizational Dynamics*, 16(4):5–21, 1988.

[HBS73] C.E. Hewitt, P. Bishop, and R. Steiger. A Universal Modular Actor Formalism for Artificial Intelligence. In *IJCAI*, pages 235–245, 1973.

[Hea] M. Heath. Performance comparison of Apache MINA and JBoss Netty. `http://blog.toadhead.net/index.php/2009/03/03/performance-comparison-of-apache-mina-and-jboss-netty/`. Accessed on: 15th September 2010.

[HH95] E. Hutchins and B. Hazlehurst. How to invent a lexicon: The development of shared symbols in interaction. In N. Gilbert and R. Conte, editors, *Artificial Societies: The Computer Simulation of Social Life*, pages 157–189. UCL Press: London, 1995.

[HHC09] Brian Heath, Raymond Hill, and Frank Ciarallo. A Survey of Agent-Based Modeling Practices (January 1998 to July 2008). *Journal of Artificial Societies and Social Simulation*, 12(4):9, 2009.

[Hic] R. Hickey. ants.clj (Clojure ants simulation). `http://clojure.googlegroups.com/web/ants.clj`. Accessed on: 15th September 2010.

[Hic10a] Rich Hickey. Clojure. `http://clojure.org/`, 2010. Accessed on: 06th April 2010.

[Hic10b] Rich Hickey. Clojure - agents. `http://clojure.org/agents`, 2010. Accessed on: 06th July 2010.

[HJF97] P.T. Hraber, T. Jones, and S. Forrest. The Ecology of Echo. *Artificial Life*, 3(3):165–190, 1997.

[HJV09] Gert Jan Hofstede, Catholijn M. Jonker, and Tim Verwaart. Simulation of effects of culture on trade partner selection. In Cesreo Hernndez, Marta Posada, and Adolfo Lpez-Paredes, editors, *Artificial Economics*, volume 631 of *Lecture Notes in Economics and Mathematical Systems*, pages 257–268. Springer Berlin Heidelberg, 2009.

[Hof01] G. Hofstede. *Culture's Consequences*. Sage Publications, 2001.

[HS97] M.N. Huhns and M.P. Singh. The agent test. *IEEE Internet Computing*, 1(5):78–79, 1997.

[HS98] M.N. Huhns and M.P. Singh. *Readings in Agents*, chapter Agents and Multiagent systems: Themes, Approaches and Challenges, pages 1–23. Morgan Kaufman, 1998.

[HTW04] A. Helsinger, M. Thome, and T. Wright. Cougaar: A Scalable, Distributed Multi-Agent Architecture. *IEEE*, 2004.

[Huh09] M. N. Huhns. From DPS to MAS to: continuing the trends. In C. Sierra, C. Castelfranchi, K.S. Decker, and J.S. Sichman, editors, *AAMAS (1)*, pages 43–48. IFAAMAS, 2009.

[IBMa] IBM SPSS Modeler Professional. `http://www.spss.com/software/modeling/modeler-pro/`. Accessed on: 15th September 2010.

[IBMb] Java theory and practice: Concurrent collections classes. `http://www.ibm.com/developerworks/java/library/j-jtp07233.html`. Accessed on: 15th September 2010.

[IEE] IEEE-FIPA (Pre-)Meeting at AAMAS 2005-07-27. `http://www.fipa.org/subgroups/ROFS-SG-docs/FIPA-Review-At-AAMAS-2005-07.htm`. Accessed on: 15th September 2010.

[Inc] Incanter: Statistical Computing and Graphics Environment for Clojure. `http://incanter.org/`. Accessed on: 15th September 2010.

[JaC] JaCa-Android. `http://jaca-android.sourceforge.net/`. Accessed on: 15th September 2010.

[JADa] Does JADE really comply with FIPA? Accessed on: 15th September 2010.

[Jadb] Jadex BDI Agent System. `http://jadex-agents.informatik.uni-hamburg.de/xwiki/bin/view/About/Overview`. Accessed on: 15th September 2010.

[JAD09] JADE - Java Agent DEvelopment Framework. `http://jade.tilab.com`, October 2009. Accessed on: 15th September 2010.

[Jasa] Jason: a Java-based interpreter for and extended version of AgentSpeak. `http://jason.sourceforge.net/Jason/Jason.html`. Accessed on: 15th September 2010.

[JASb] Java Agent Services. `http://sourceforge.net/projects/jas/`. Accessed on: 15th September 2010.

[JASc] Journal of Artificial Societies and Social Simulation. `http://jasss.soc.surrey.ac.uk`. Accessed on: 15th September 2010.

[Java] Java SE Downloads. `http://www.oracle.com/technetwork/java/javase/downloads/index.html`. Accessed on: 15th September 2010.

[Javb] Micro-agents and PRS. `http://sourceforge.net/projects/javaprs/`. Accessed on: 15th October 2010.

[JBo] JBoss. `http://www.jboss.org/`. Accessed on: 15th September 2010.

[Jet] Jetlang. `http://code.google.com/p/jetlang/`. Accessed on: 25th July 2010.

[JSR] JSR 87: Java Agent Services. `http://jcp.org/en/jsr/detail?id=087`. Accessed on: 25th July 2010.

[JW00] N.R. Jennings and M. Wooldridge. Agent-Oriented Software Engineering. *Artificial Intelligence*, 117:277–296, 2000.

[KNI] KNIME — Konstanz Information Miner. `http://www.knime.org/`. Accessed on: 25th October 2010.

[Kni86] Thomas F. Knight. An architecture for mostly functional languages. In *LISP and Functional Programming*, pages 105–112, 1986.

[Kor] Korus. `http://code.google.com/p/korus/`. Accessed on: 25th July 2010.

[KP98] Franziska Klügl and Frank Puppe. The Multi-Agent Simulation Environment SeSAm, April 1998.

[KSA09] Rajesh K. Karmani, Amin Shali, and Gul Agha. Actor Frameworks for the JVM Platform: A Comparative Analysis. In *7th International Conference on the Principles and Practice of Programming in Java*, 2009.

[LCG] Linear congruential generator. `http://en.wikipedia.org/wiki/Linear_congruential_generator`. Accessed on: 15th September 2010.

[LCRP+05] Sean Luke, Claudio Cioffi-Revilla, Liviu Panait, Keith Sullivan, and Gabriel Balan. MASON: A Multiagent Simulation Environment. *Simulation*, 81(7):517–527, July 2005.

[Leea] T. Lee. Netty - the Java NIO Client Server Socket Framework. `http://jboss.org/netty`. Accessed on: 15th September 2010.

[Leeb] T. Lee. Performance Comparison between NIO Frameworks. `http://gleamynode.net/articles/2232/`. Accessed on: 15th September 2010.

[Luk] S. Luke. Sean Luke: Code. `http://cs.gmu.edu/~sean/research/`. Accessed on: 15th September 2010.

[MÖ90] M. Möhring. *MIMOSE. Eine funktionale Sprache zur Beschreibung und Simulation individuellen Verhaltens interagierender Populationen.* PhD thesis, Universität Koblenz, 1990.

[Mas43] A.H. Maslow. A Theory of Human Motivation. *Psychological Review*, 50(4):370–396, 1943.

[MBB+98] D.S. Milojicic, M. Breugst, I. Busse, J. Campbell, S. Covaci, B. Friedman, K. Kosaka, D.B. Lange, K. Ono, M. Oshima, C. Tham, S. Virdhagriswaran, and J. White. MASIF: The OMG mobile agent system interoperability facility. *Personal and Ubiquitous Computing*, 2(2), 1998.

[MCd96] B. Moulin and B. Chaib-draa. *Foundations of Distributed Artificial Intelligence*, chapter An Overview of Distributed Artificial Intelligence, pages 3–55. Wiley, 1996.

[McS02] B. McSweeney. Hofstede's Model Of National Cultural Differences And Their Consequences: A Triumph Of Faith - A Failure Of Analysis. *Human Relations*, 55(1):89118, 2002.

[MDSC02] M.B. Marietto, N. David, J.S. Sichman, and H. Coelho. Requirements analysis of agent-based simulation platforms: State of the art and new prospects. In *Multi-Agent-Based Simulation II*, Berlin, Heidelberg, New York, 2002. Third International Workshop, MABS 2002 Bologna, Italy, July 2002, Springer.

[MFD09] F. Michel, J. Ferber, and A. Drogoul. *Multi-Agent Systems: Simulation and Applications*, chapter Multi-Agent Systems and Simulation: A Survey from the Agent Community's Perspective, pages 3–52. CRC Press, 2009.

[MGF04] F. Michel, A. Gouaïch, and J. Ferber. *Multi-Agent-Based Simulation III*, volume LNCS 2927, chapter Weak interaction and strong interaction in agent-based simulations, pages 43–56. Springer, 2004.

[MK06] J. Magee and J. Kramer. *Concurrency: State models and Java programming*. Wiley, second edition, 2006.

[MM05] Vladimir Marik and Duncan McFarlane. Industrial adoption of agent-based technologies. *IEEE Intelligent Systems*, 20:27–35, 2005.

[MMR92] D.H. Meadows, D.L. Meadows, and J. Randers. *Beyond the Limits*. Chelsea Green, 1992.

[MN98] M. Matsumoto and T. Nishimura. Mersenne Twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. on Modeling and Computer Simulation*, 8(3), 1998.

[MPT95] J.P. Müller, M. Pischel, and M. Thiel. Modelling reactive behaviour in vertically layered agent architectures. *Intelligent Agents: Theories, Architectures and Languages*, LNAI 890, Springer:261–276, 1995.

[MRBCL96] Nelson Minar, Y. Roger Burkhart, and Z. Chris Langton. The Swarm Simulation System: A Toolkit for Building Multi-Agent Simulations. `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.108.1436`, 1996.

[MSK+07] T. Malsch, C. Schlieder, P. Kiefer, M. Lbcke, R. Perschke, M. Schmitt, and K. Stein. Communication between process and structure: Modelling and simulating message reference networks with COM/TE. *Journal of Artificial Societies and Social Simulation*, 10(1), 2007.

[Mue96] J. Mueller. The Design of Intelligent Agents: a layered approach. *Lecture Notes in Computer Science*, 1177, 1996.

[NBPC01] M. Nowostawski, G. Bush, M. Purvis, and S. Cranefield. A Multi-Level Approach and Infrastructure for Agent-Oriented Software Development. Technical report, Department of Information Science, University of Otago, 2001.

[NCV06] M. North, N. Collier, and J. Vos. Experiences creating three implementations of the RePast agent modelling toolkit. *ACM Transactions on Modelling and Computer Simulation*, 16(1):1–25, 2006.

[Neg97] N. Negroponte. *Software Agents*, chapter Agents: From Direct Manipulation to Delegation, pages 57–66. AAAI Press, 1997.

[NET] .NET Framework Developer Center. `http://msdn.microsoft.com/en-us/netframework/default.aspx`. Accessed on: 15th September 2010.

[Nex10] ISO/IEC JTC1 SC22 WG21 N3092: Programming Languages - C++ (Draft). `http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2010/n3092.pdf`, March 2010. Accessed on: 15th September 2010.

[NP07] M. Nowostawski and M. Purvis. The Concept of Autonomy in Distributed Computation and Multi-Agent Systems. Information Science Discussion Paper 2007/06, Dept. of Information Science, University of Otago, Dunedin, New Zealand, 2007.

[NPC01] M. Nowostawski, M. Purvis, and S. Cranefield. KEA - Multi-Level Agent Architecture. Technical report, Department of Information Science, University of Otago, 2001.

[NPT10] C. D. Nguyen, A. Perini, and P. Tonella. Goal-oriented testing for MASs. *International Journal for Agent-Oriented Software Engineering*, 4(1):79–109, 2010.

[OCC88] A. Ortony, G. Clore, and A. Collins. *The cognitive structure of emotions*. Cambridge University Press, 1988.

[Ode02] James Odell. Objects and Agents Compared. *Journal of Object Technology*, 1(1):41–53, 2002.

[Ous96] J.K. Ousterhout. Why threads are a bad idea (for most purposes). In *Invited Talk, USENIX 1996*, June 1996.

[Par99] H. Van Dyke Parunak. "go to the ant": Engineering principles from natural multi-agent systems, 24th March 1999.

[PBL03] A. Pokahr, L. Braubach, and W. Lamersdorf. Jadex: Implementing a BDI-infrastructure for JADE Agents. *EXP - In Search of Innovation*, 3(3):76–85, 2003. Accessed on: 10th October 2009.

[PTB+06] M. Pechoucek, S.G. Thompson, J.W. Baxter, G.S. Horn, K. Kok, C. Warmer, R. Kamphuis, V. Maric, P. Vrba, K.H. Hall, F.P. Maturana, K. Dorer, and M. Calisti. Agents in industry: the best from the aamas 2005 industry track. *Intelligent Systems, IEEE*, 21(2):86 – 95, 2006.

[Rao96] A.S. Rao. Agentspeak(l): Bdi agents speak out in a logical computable language. In *Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-96)*, 1996.

[Rei] M. Reinhold. JSR 51: New I/O APIs for the Java Platform. `http://www.jcp.org/en/jsr/detail?id=51`. Accessed on: 25th July 2010.

[RG95] Anand A. Rao and Michael P. Georgeff. BDI Agents: From Theory to Practice. *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, 1995.

[RLJ06] Steven F. Railsback, Steven L. Lytinen, and Stephen K. Jackson. Agent-based Simulation Platforms: Review and Development Recommendations. *Simulation*, 82(9):609–623, 2006.

[RW73] H. Rittel and M. Webber. Dilemmas in a General Theory of Planning. *Policy Sciences*, 4:155169, 1973.

[SAS] SAS Enterprise Miner. `http://www.sas.com/technologies/analytics/datamining/miner/`. Accessed on: 15th September 2010.

[Sch71] T.C. Schelling. Dynamic Models of Segregation. *Journal of Mathematical Sociology*, 1:143–186, 1971.

[SDM10] B.R. Steunebrink, M. Dastani, and J.C. Meyer. Emotions to control agent deliberation. In v.d. Hoek, Kaminka, Lesperance, Luck, and Sen, editors, *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, 2010.

[Sea69] John R. Searle. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, 1969.

[Sea76] John Searle. The Classification of Illocutionary Acts. *Language in Society*, 5:1–24, 1976.

[SF] M. Saber and J. Ferber. MAGR: Integrating Mobility of Agents with Organizations. `http://www.madkit.net/documents/articles/MAGR_IADIS07_Mansour_Ferber.pdf`. Accessed on: 15th September 2010.

[SH99] M.P. Singh and M.N. Huhns. *Intelligent Information Agents*, chapter Social Abstractions for Information Agents, pages 37–52. Springer, 1999.

[Sho93] Y. Shoham. Agent-oriented Programming. *Artificial Intelligence*, 60(1):51–92, 1993.

[Sho97] Y. Shoham. *Software Agents*, chapter An Overview of Agent-Oriented Programming, pages 271–290. AAAI Press, 1997.

[SHYN06] Weiming Shen, Qi Hao, Hyun Joong Yoon, and Douglas H. Norrie. Applications of agent-based systems in intelligent manufacturing: An updated review. *Advanced Engineering Informatics*, 20(4):415 – 431, 2006.

[Sim96] H.A. Simon. *The Sciences of the Artificial*. MIT Press, 1996.

[Sin98] Munindar P. Singh. Agent Communication Languages: Rethinking the Principles. *Computer*, 31(12):40–47, 1998.

[SM08] Sriram Srinivasan and Alan Mycroft. Kilim: Isolation-Typed Actors for Java. In *European Conference on Object Oriented Programming ECOOP 2008, Cyprus*, 2008.

[Sma98] ANSI Smalltalk Standard . `http://www.smalltalk.org/versions/ANSIStandardSmalltalk.html`, 1998. Accessed on: 15th September 2010.

[Smi80] R.G. Smith. The Contract Net Protocol. *IEEE Transactions on Computers*, C-29(12), 1980.

[Squ08] F. Squazzoni. The Micro-Macro Link in Social Simulation. *Sociologica*, (1), 2008.

[ST97] N. Shavit and D. Touitou. Software Transactional Memory. *Distributed Computing*, 10(2):99–116, 1997.

[ST09] L. Sterling and K. Taveter. *The Art of Agent-Oriented Modeling.* MIT Press, Cambridge, MA, 2009.

[Sut05] H. Sutter. The free lunch is over: A fundamental turn toward toward concurrency. *Dr. Dobb's Journal*, March 2005.

[Sys] Henry George Liddell, Robert Scott, A Greek-English Lexicon. `http://www.perseus.tufts.edu/hopper/text?doc=Perseus%3Atext%3A1999.04.0057%3Aentry%3Dsu%2Fsthma`. Accessed on: 15th September 2010.

[TH04] Robert Tobias and Carole Hofmann. Evaluation of free Java-libraries for social-scientific agent based simulation. *J. Artificial Societies and Social Simulation*, 7(1), 2004.

[Thr] The Thread.yield() method. `http://www.javamex.com/tutorials/threads/yield.shtml`. Accessed on: 15th September 2010.

[TMEL09] G.K. Theodoropoulos, R. Minson, R. Ewald, and M. Lees. *Multi-Agent Systems: Simulation and Applications*, chapter Simulation Engines for Multi-Agent Systems, pages 77–105. CRC Press, 2009.

[Tod83] E. Todd. *La troisieme planete: Structures familiales et systemes ideologiques.* Seuil, 1983.

[Tro97] K.G. Troitzsch. *Simulating Social Phenomena*, chapter Social Sciences Simulation. Origins, Prospects, Purposes, pages 41–54. Springer, 1997.

[Tro04] K.G. Troitzsch. *18th European Simulation Multiconference. Networked Simulations and Simulation Networks*, chapter Validating simulation models, pages 265–270. The Society for Modeling and Simulation International, SCS Publishing House, 2004.

[Tro09a] K.G. Troitzsch. *Multi-Agent Systems: Simulation and Applications*, chapter Multi-Agent Systems and Simulation: A Survey from an Application Perspective, pages 53–75. CRC Press, 2009.

[Tro09b] Klaus G. Troitzsch. Perspectives and challenges of agent-based simulation as a tool for economics and other social sciences. In Carles Sierra, Cristiano Castelfranchi, Keith S. Decker, and Jaime Simão Sichman, editors, *AAMAS (1)*, pages 35–42. IFAAMAS, 2009.

[vB68] L. von Bertalanffy. *General System Theory: Foundations, Development, Applications*. George Braziller, 1968.

[vDP00] H. van Dyke Parunak. Agents in Overalls: Experiences and Issues in the development and deployment of industrial agent-based systems. *International Journal of Cooperative Information Systems*, 9(3):209–227, 2000.

[Vir] VirtualBox. `http://www.virtualbox.org/`. Accessed on: 15th September 2010.

[vtV] Remco van 't Veer. clj-android. `http://github.com/remvee/clj-android`. Accessed on: 08th October 2010.

[War79] M.W. Wartofsky. *Models*. D. Reidel, Dordrecht, 1979.

[WD04] H. Weigand and V. Dignum. I am autonomous, you are autonomous. In M. Nickles, M. Rovatsos, and G. Wei, editors, *Agents and computational autonomy: Potential, Risks, and Solutions*, volume 2969 of *LNCS*, pages 227–236. Springer, 2004.

[WG06] N. Weidmann and L. Girardin. GROWLab: A Toolbox for Social Simulation. *First World Congress on Social Simulation*, 2006.

[Whi97] J.E. White. *Software Agents*, chapter Mobile Agents, pages 437–472. AAAI Press, 1997.

[Wik] Parallel computing. `http://en.wikipedia.org/wiki/Parallel_computing`. Accessed on: 15th September 2010.

[Wil99] U. Wilensky. NetLogo. `http://ccl.northwestern.edu/netlogo/`, 1999. Accessed on: 15th September 2010.

[Wim] T. Wimberly. JIT compiler coming to Android sooner than you think? `http://androidandme.com/2010/02/news/jit-compiler-coming-to-android-sooner-than-you-think/`. Accessed on: 08th October 2010.

[Win05] M. Winikoff. Jack intelligent agents: An industrial strength platform. In Gerhard Weiss, Rafael Bordini, Mehdi Dastani, Jrgen Dix, and Amal Fallah Seghrouchni, editors, *Multi-Agent Programming*, volume 15 of *Multiagent Systems, Artificial Societies, And Simulated Organizations*, pages 175–193. Springer US, 2005.

[Win09] M. Winikoff. Future directions for agent-based software engineering. *International Journal for Agent-Oriented Software Engineering*, 3(4):402–410, 2009.

[Wit21] L. Wittgenstein. *Tractatus Logico-Philosophicus*. Annalen der Naturphilosophie, 1921.

[WJ95] M.J. Wooldridge and N.R. Jennings. Agent Theories, Architectures, and Languages: A Survey. In *Intelligent Agents: ECAI-94 Workshop on Agent Theories, Architectures, and Languages*, 1995.

[WJK00] Michael Wooldridge, Nicholas R. Jennings, and David Kinny. The Gaia Methodology for Agent-Oriented Analysis and Design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, September 2000.

[Woo97] Michael Wooldridge. Agent-based Software Engineering. *IEE Proceedings in Software Engineering*, 144:26–37, 1997.

[Woo09] M. Wooldridge. *An Introduction To MultiAgent Systems*. Wiley & Sons, 2009.

[XSt] XStream. `http://xstream.codehaus.org/`. Accessed on: 15th October 2010.

[Zei76] B. Zeigler. *Theory of Modeling and Simulation*. Wiley Interscience, 1976.

[ZEU] ZEUS Agent Toolkit. `http://sourceforge.net/projects/zeusagent/`. Accessed on: 15th September 2010.