Evaluation of Agent–Oriented Software Methodologies – Examination of the Gap Between Modeling and Platform

Jan Sudeikat[1], Lars Braubach[2], Alexander Pokahr[2], and Winfried Lamersdorf[2]

[1] University of Applied Sciences Hamburg,
Berliner Tor 3, 20099 Hamburg, Germany,
Jan.Sudeikat@hamburg.de
[2] Distributed Systems and Information Systems,
Computer Science Department, University of Hamburg,
Vogt–Kölln–Str. 30, 22527 Hamburg, Germany
Tel. +49-40-42883-2091
{braubach | pokahr | lamersd}@informatik.uni-hamburg.de

Abstract. More and more effort is made to provide methodologies for the development of agent–based systems. Awareness has grown that these are necessary to develop high quality agent systems. In recent years a number of proposals has been given. Based on our experiences we argue that a complete evaluation of methodologies cannot be done without considering target platforms, because the differences between available implementations are too fundamental to be ignored. In order to conduct a suitable comparison we present a flexible evaluation framework that takes platform specific criteria into account. Part of this framework is a procedure to derive relevant criteria from the evaluated platforms and methodologies. In combination with a set of platform dependent and independent criteria our framework allows evaluation of the appropriateness of methodologies with respect to platforms. As a consequence, also the suitability of methodologies for an individual platform, or vice versa of several platforms for an individual methodology can be examined. To show the usefulness of our proposal, we evaluate the suitability of different methodologies for an example platform.

1 Introduction

Besides the necessity for reliable agent platforms the need for the methodical development of applications has been noticed (as described in [18], [19]) and is addressed by a number of proposed methodologies for building agent–based software applications (surveyed in [36], [16]). According to [31] a methodology aids development through (1) guidance by a life cycle process, (2) a set of predefined techniques (guidelines, heuristics, etc.) and (3) allows modeling by providing a suitable notation. What these three elements comprise is different in the specific proposals. This makes it difficult for organizations to decide which one to use. The selection of the right methodology is crucial for the success of a large

software project, because developers will need guidance how to use this new paradigm. According to [21], suitable methodologies are a key factor to introduce agent–orientation as an engineering approach to the industry. Since a number of methodologies has been proposed there is a need for structured means to select appropriate ones with respect to a concrete setting. Organizations will need guidance to select one to adopt for their own development.
To address these needs different frameworks for comparison have been proposed. These use feature–based evaluations of numerous criteria to identify superior ones. However none of the proposed approaches takes the target agent platform into account. Platforms imply different concepts of agency in different peculiarities. Since the support for agent oriented concepts differs between concrete implementations, our proposal stresses that there are both platform dependent and independent criteria to evaluate. In the following pages we identify a set of aspects to take into account and show how to derive platform dependent criteria from target platforms.
To guide evaluation regarding these criteria we present a flexible framework to examine the match between platform and methodology. It allows to examine the appropriateness of methodologies for a preselected platform or vice versa the appropriateness of different platforms for an individual methodology. This flexibility is advantageous, because software producing organizations are seldom free to choose tools and methodologies as they like. Often there will be restrictions, e.g. industry projects may have to focus on a single platform because a client demands its usage or universities may favor a special methodology and need to find a suitable software environment.
The usage of our approach is clarified by an example evaluation of the appropriateness of prominent methodologies for a concrete platform, implementing the BDI architecture [28]. Following our framework we derive a number of platform dependent criteria from this implementation and together with platform independent criteria we examine the match between pairs of platform and methodology.

The next section gives a brief overview of proposed methodologies and examines the comparisons of agent–oriented methodologies that have already been conducted. Section three presents our framework. First we describe the modus operandi of our approach and present thereafter the set of criteria to examine. The fourth section shows an example evaluation of a set of methodologies for a concrete platform. Finally we conclude and give prospects for further work.

2 Background

In [14] a number of different methodologies have been arranged in a genealogy. Figure 1 gives an overview of current proposals in a similar way. It illustrates the main influences on the individual methodologies. We found Object–Orientation (OO), Knowledge Engineering (KE) and Requirements Engineering (RE) as ancestors, which have been extended by agent programming abstractions (denoted

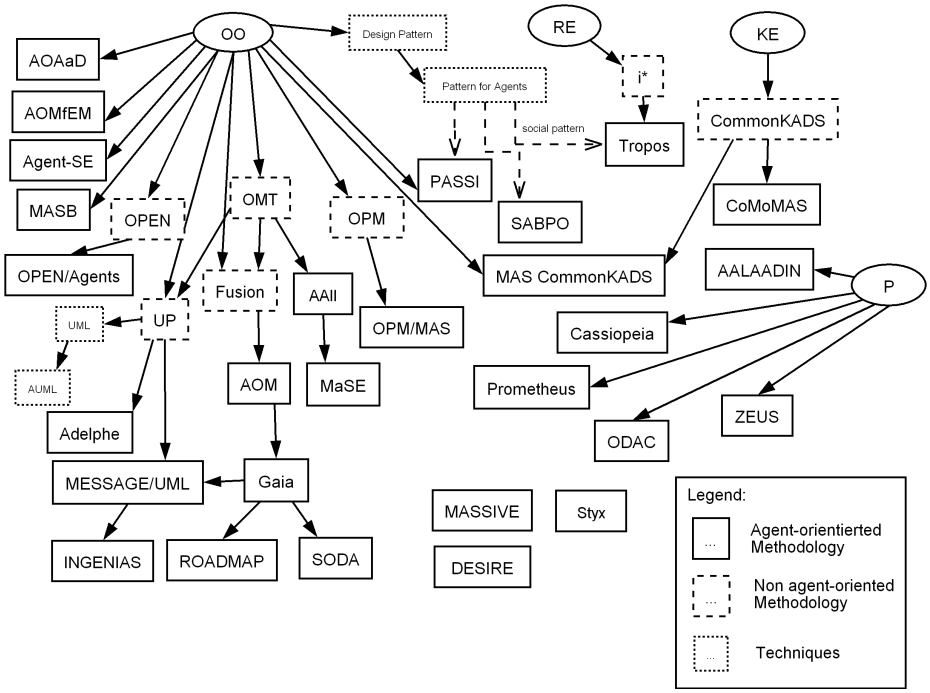

Fig. 1. Genealogy of proposed methodologies

by ovals). P denotes a coarse category of sources. Associated methodologies were inspired by experiences with specific agent platforms or architectures. Intermediate forms (dotted rectangles) have been extended to truly agent–oriented methodologies (rectangles). A complete list of references to these methodologies is omitted here for sake of brevity, it can be found in [34].

A small amount of work has been conducted to compare agent–oriented methodologies. These approaches found two sources of features to examine. First, they adopt general software–engineering criteria, which have been found relevant for evaluations of methodologies according to various paradigms. Secondly, they identified specific criteria that are needed to support agent–oriented concepts in development. All of them gather a set of criteria, which is supposed to be independent from the field of application or platform. They point out what is needed for a comprehensive methodology together with individual drawbacks and advantages.

O'Malley and DeLoach [22] collected a set of criteria to guide organizations in deciding, whether an AOSE methodology should be adopted or an object–oriented methodology is appropriate for a particular project. They distinguish between *Management Issues* and *Project Requirements*. The criteria have been validated

by a survey [22]. Management issues examine the consequences an adoption of a methodology causes for the (software producing) organization. Aspects like cost and suitability for the organization are examined. Project requirements concentrate on the technical issues. The authors found a set of aspects that need to be modeled (like interactions, distribution, etc.). These criteria are rated and a weighted mean value is calculated. Kitchenham [20] describes the vagueness and shortcomings of these approaches.

Cernuzzi and Rossi [6] proposed a qualitative analysis followed by a quantitative rating. They constructed a so called *Attributes Tree*, which organizes the found criteria in weighted branches. After rating the leafs the value of the root can be calculated and compared to other methodologies.
The authors identified three kinds of criteria. *Internal Attributes* characterize the internal structures of the agents, *Interaction Attributes* describe how the interactions inside the system can be modeled. Finally, the *Other Process Requirements* judge the design and development–process, proposed by the methodologies.

The above described proposals compared the methodologies by a screening of the criteria, which should be compared as unbiased as possible. Dam and Winikoff [8] evaluated methodologies by surveying inventors of the selected methodologies and developers, who modeled a case study. They divided their found criteria into *Concepts & Properties*, *Modeling & Notation*, *Processes* and *Pragmatics*.
The concepts and processes are suitable for a feature–based analysis. They examine the extent to which specific concepts are supported and the coverage of different stages in development. Besides requirements, architectural/detailed design, implementation and testing, these also include deployment and maintenance. The other two categories examine the notation and the general management and technical characteristics of the methodologies. Judging the appropriateness of a notation and the mentioned characteristics is a difficult task. Dam and Winikoff successfully addressed it by a survey approach.

Shehory and Sturm [32] developed a catalog of criteria for feature–based analysis of AOSE methodologies. They distinguished between *Software–Engineering Criteria* and *Agent–Based Characteristics*. Their performed analysis identified areas of improvement, to suit the needs of developers.
In [33] they recently adopted the classification from [8] and combined the criteria from this work with their own. They give a sound overview of what a mature methodology should offer in the field of agent–oriented software engineering. Their set of criteria is suitable to show the drawbacks of currently proposed methodologies and therefore gives suggestions for further development.

3 A Platform Dependent Comparison Framework

All of the approaches characterized in section 2 focus on the identification of a superior methodology among a group of candidates. These are influenced by

previous work examining methodologies of various paradigms. The spread of e.g. object oriented methodologies was possible, because there was a sound understanding of the object–oriented paradigm in itself. For agent systems we lack this sound foundation to build methodologies upon. Even if we look at a particular architecture - like BDI - we still see a lot of differences between the available implementations and theories. These denote conceptual differences in the expressiveness for the properties of individual agents. We conclude that, in opposition to other software engineering paradigms, which may be language independent, the use of agent–oriented methodologies is to some degree dependent from the used platform. Different implementations of agent architectures need different levels of expressiveness. The suitability of a methodology is highly influenced by the support for this expressiveness.
Having noted this, we present a framework to evaluate the appropriateness of methodologies in relation to platforms. Figure 2 gives a visual description of our approach. Two kinds of criteria are evaluated. Platform independent criteria can

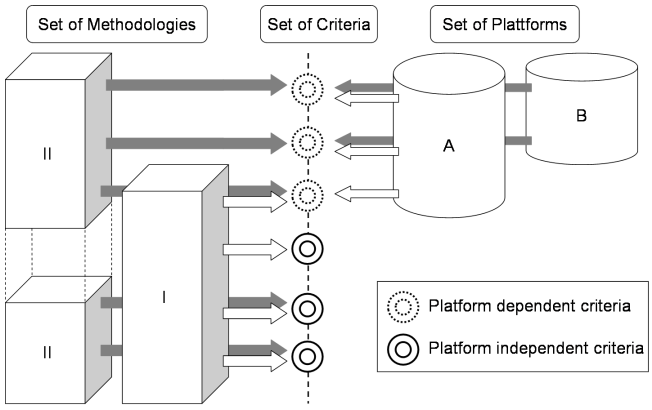

Fig. 2. Modus Operandi for the Evaluation Framework

be examined in a feature analysis, for platform dependent ones the properties of the implemented concepts need to be compared to properties supported in the methodology. The match between them is examined to show their appropriateness. As Figure 2 depicts, not all features of a methodology will match properties of all platforms. For our purposes it is important to identify the differences. The shared absence of properties is regarded as a match, since it also identifies appropriateness.
Examining differences in this way allows an important adjustment to evaluation for different purposes. As stated earlier, software producing organizations are seldom free to choose tools and methodologies unbiased. Industry projects may have to focus on a single platform because their client demands a dedicated platform, universities may favor a special methodology and need to find a suitable

software environment. Premises of this kind are regarded by our approach. There are three possible relations for evaluation. Evaluations of one methodology to many platforms (1:n), several methodologies for one specific platform (n:1; see the later given example) and a n:m scenario where n methodologies are rated to m platforms (see Figure 2). All of these can be conducted in the same modus operandi. Evaluations of the latter relations force the conductor to deliberate about match vs. quality of the methodology. A good matching methodology may not comprise all the platform independent criteria. Desirable is a methodology that matches a given platform and supports as many of the independent criteria as possible.

3.1 Criteria

Before an examination following our approach can take place the relevant criteria need to be identified. The selection is based on the above described directions of comparison, but we have found a set of aspects every evaluation has to take into account. For our framework we adopted the classification of these criteria from [8]. Therefore, the criteria are separated into four groups. *Concepts*, *Notation*, *Process* and *Pragmatics*. Table 1 lists our selection of criteria.[3] The concepts

1st Phase: screening		2nd Phase: examination	
Concepts	Notation	Process	Pragmatics
Internal Architecture*	Usability	Coverage of Workflows	Tool Support
Social Architecture*	Expressiveness	Management	Connectivity*
Communication*	Refinement	Complexity	Documentation
Autonomy	Dependency of Models	Properties of Process	Usage in Projects
Pro–activity	Traceability		
Distribution*	Clear definition		
	Modularity		

Table 1. Overview of the evaluated criteria[3]

are mostly platform dependent, because even for a concrete agent architecture (e.g. BDI), there is no sound foundation for the properties of the single concepts. For example the representation and expressiveness of goals varies greatly among different implementations. The notation and process are independent from the platform. These aspects describe the usage of a methodology. Also the pragmatics concentrate on an examination of the support for a methodology. The following sections will give a further explanation of the listed criteria.

The Agent–Oriented Concepts These Criteria have to be supported by suitable methodologies.

[3] Platform dependent criteria are marked with an asterisk.

Internal Architecture* The concepts which describe the internals of an agent vary greatly between the proposed architectures. For example, BDI architectures [13] describe agents with notions of mental attitudes, other approaches use different internal representations (e.g. Subsumption-architecture [3], Soar [35]).

Social Architecture* These concepts describe, which social models are used to organize the multi–agent systems. Prominent models are the notions of *groups*(e.g. AALAADIN [10]) or *teams* [15], others allow agents to offer services, e.g. via a yellow pages directory [11].

Communication* Different Communication models have been proposed. Prominent models are message based (e.g. speech acts using ontologies [12] or event based message exchange), others use memory based (e.g. blackboard [7]) architectures.

Autonomy The abilities of an agent to solve problems in an autonomous way, is illustrated by the modeling of functionalities or tasks an agent can execute on its own authority. In addition, it is helpful to express the mechanisms used to make decisions about which actions to take.

Pro–activity It is needed to express the proactive abilities of a agent.

Distribution* It is desirable to be able to express the allocation of agents to places in the environment.

The Notation The notation defines abstract views on the most important aspects of the developed system. It consists of symbols, syntax and semantic. The **Usability** is supported by a clearly defined and intuitively comprehensible notation that is easy to draw. To support both the requirements analysis, the analysis and design of the system to build, an **expressive** notation supports several views on the system to develop. It allows to express the functional, structural and dynamic properties, where the structure includes the data and flow of data inside the system.
Furthermore these models should support some technical criteria to allow convenient usage. During development the **refinement** and **modularity** of the single models should be supported. Models should **depend** on each other and single artifacts should be **traceable**. It is indispensable that syntax and semantic are **clearly defined**.

The Process To evaluate the proposed development processes we compare them to the "Unified Process" (UP) [17]. In [33] it is already proposed to evaluate the **coverage** of the 5 basic workflows from the UP. The selection of the UP is arbitrary. It is suitable as a well known reference to ease comparison. Many more complete methodologies have been developed (e.g. the Rational Unified Process [29]), but an illustration of the coverage of these 5 basic activities is suitable, due to the immaturity of current proposals. The individual workflows are: *Requirements* (gathering and documentation of necessities), *Analysis* (further examination of the problem domain), *Design* (defining how the software will be implemented), *Implementation* (conversion of design into executables)

and *Testing* (development of test–cases, their execution and debugging). Furthermore we consider the support for the **management** of an agent–oriented software project. Currently this support mainly consists of heuristics and guidelines. The **complexity** of the process measures the necessary effort to learn and use it. Favorable is one where the tasks of the single development steps and the sequence of them is easy to understand and to comprehend. Finally, **properties of process** note special properties, e.g. whether it is an iterative approach or not, top–down or bottom–up, etc.

The Pragmatics The pragmatics are dominated by the impressions of the available (CASE–)**tool support**. Evaluating these tools is a difficult task in itself. Their usability is influenced by many aspects, in our evaluation we tried to take ergonomic aspects into account. Tools should be easy to use and support the whole development cycle. The currently available tools allow drawing the notations and checks for consistency. The **connectivity** describes the platform dependent aspects of the tool support. When tool support is available it is desirable to have a connection between the design artifacts and the target platform to use it seamlessly in development. In the most convenient case there will be the possibility to directly generate code for target platforms. Another important aspect is the available **documentation**. This has a great impact on the usability and understanding of a methodology. Also the documentation of the tools is important here. Reported experiences with the **usage in projects** are an important factor for judging the maturity of a methodology.

3.2 Evaluation Process

The actual evaluation takes place in two steps. First, the abstract concepts set need to be concretized with respect to methodologies and target platform(s). Then they can be examined together with our proposed platform independent criteria.
In opposition to the properties of notations and pragmatics, the support of concepts and the properties of a proposed process can be examined by a simple screening of the single methodologies. Evaluating the notations and pragmatics is a difficult task in itself. It has to take objective criteria as well as ergonomic aspects into account. Also the usability of CASE–Tools (an important part of the pragmatics) is influenced by these. The work of Wood et al. [37] gives hints how to examine a notation and Kitchenham [20] is also taking the examination of tools into account. But differences are subtle and subjective to the conductor screening specific candidates. For our example we made a case study to evaluate these. Other possibilities are formal experiments or surveys according to the available resources and purposes.

4 Example Evaluation

To give an example for an evaluation following our framework we will evaluate how the methodologies *MaSE* [9], *Tropos* [2] and *Prometheus* [24] (see Figure 1)

match up the *Jadex* [26], [1] agent platform. Jadex, developed at the Distributed Systems and Information Systems unit at the Computer Science Department of the University of Hamburg, is an add–on to the popular JADE[4] agent platform. It extends JADE with sophisticated BDI mechanisms and is is under busy development.

The Multiagent Systems Engineering Methodology (MaSE) proposes a complete life cycle methodology for multi–agent systems. It guides the developer from the specification of the system to the final implementation. Agents are described as finite state machines.
Tropos has been influenced by the *i** framework from Yu [39] for analysis of the *early requirements* of a software system. It leads the developers to an understanding of an agent–oriented system as an organization of actors. These seek to achieve goals by means of plans and have dependencies to other actors.
Prometheus has been influenced by the JACK[5] agent platform. Static structures of multi agent systems are clearly illustrated. This methodology allows very detailed modeling by descriptors, which hold design specific properties.
These preselected methodologies are well documented, most mature, support BDI–concepts and CASE–tools are available (in [8] the same selection has been found).

We are here evaluating the suitability for only one platform. Note that this forms an *3:1* relation (three methodologies are rated to one platform). According to the modus operandi we first derive the relevant platform dependent criteria from the target platform. Thereafter the concepts and process are rated by a screening and finally the notations and tools are evaluated using a case study. The presented considerations are based on our own evaluation of a suitable methodology for this BDI platform, which has been conducted in the context of a diploma thesis [34].

4.1 Selection of Criteria

Considering only one platform makes it fairly easy to identify the internal, social architectures and the communication concepts. Finding the relevant criteria means to examine the platforms and methodologies to find aspects of the platforms, which need to be supported by methodologies and vice versa. For our example evaluation we found:

- **Internal Architecture:**
  Goals, Plans, Beliefs (BDI–Architecture) Since Jadex is focused on the use of BDI–concepts, these have to be supported by the methodology. It is needed to be able to describe how goals (by which plans) can

[4] http://sharon.cselt.it/projects/jade/
[5] http://www.agent-software.com

be achieved and which beliefs these plans need to access. Properly described elements allow appropriate modeling of properties to implement using Jadex.

Capabilities This is a concept to unitize BDI systems into functional modules, as described in [4].

Events These express the reactiveness of agents. It should be possible to model changes in the environment of single agents by events. Jadex also supports internal events to express changes inside an agent. The kind of event should be exactly describable to allow inferring the filters Jadex uses to distinguish events.

– **Social Architecture:**

Roles Some methodologies use Roles as a concept to structure a multi agent system and to identify single agent classes. Therefore, we need to examine how Jadex supports their implementation.

– **Communication:**

Protocols These characterize the communication between the agents and are supported by the underlying JADE platform.

Messages In Jadex the exchanged messages follow the FIPA model of agent communication [12].

4.2 Examining the Concepts and Process

Table 2 summarizes our results for the concepts and processes. All three methodologies develop a system coming from the identified goals. As opposed to MaSE, Tropos and Prometheus use the BDI–notions throughout the whole development cycle. In all methodologies, the modeled goal concepts differ from the ones

		MaSE	Tropos	Prometheus
Concepts:				
Internal Architecture	Goals*	+|≠|+	+|≈|+	+|≠|+
	Plans*	+|≈|+	+|≈|+	+|≈|+
	Beliefs*	+|≈|+	+|≈|+	+|≈|+
	Capabilities*	−|≠|+	+|≈|+	+|=|+
	Events*	+|≈|+	+|≈|+	+|≈|+
Social Architecture	Roles*	+|≠|−	+|≠|−	−|=|−
Communication	Protocols*	+|≠|−	+|≠|−	+|≠|−
	Messages*	+|≈|+	+|≈|+	+|≈|+
	Autonomy	++	++	++
	Pro–activity	+	++	++
	Distribution*	+|=|+	−|≠|+	−|≠|+
Process:				
	Coverage of Workflows	3/5	4/5	4/5
	Management	n.a.	n.a.	n.a.
	Complexity	++	++	++
	Properties of Process	for all iterative and top-down		
Support	− −: poor −: not well n.a.: not available +: well ++: very well			
Match	**Left hand side:** methodology supports property: + / - **Middle:** match between the properties no match: ≠ coarse match: ≈ good match: = **Right hand side:** platform supports property: + / -			

Table 2. Evaluation results: Concepts and Process

used in Jadex. In Prometheus and MaSE agents are associated to system goals. Tropos is more suitable, because both system and individual goals in addition to the contribution/decomposition of plans are described. Modeled plans in all methodologies lack Jadex specific properties. Only Prometheus describes the individual beliefs of agents in detail, but the structures of the beliefbases differ, causing a slight mismatch. Tropos and Prometheus support the concept of capabilities. Only in Prometheus events are stated explicitly. It is also distinguished between *percepts* (recognized changes in the environment) and the resulting relevant events (*incidents*) for the system. Jadex is not aware of this distinction. The other methodologies describe events implicit in their UML–based models for design.
Roles are supported by MaSE and Tropos. Roles are used as means to identify the different types of agents the system will be composed of. Roles are not explicitly supported by Jadex, they can be implemented using services. Since both Prometheus and Jadex don't support this specific concepts they closely match. Protocols between agents are described in Tropos and Prometheus by the sequence of transmitted messages. MaSE instead describes the exchange of messages in relation to the processing inside the agents. However these representations are of the same suitability for Jadex. In MaSE and Tropos the content of a message is not explicitly described. Only Prometheus defines special descriptors to describe properties of messages. These are not compliant to FIPA ACL messages and are therefore slightly mismatching the messages used in Jadex.
The autonomy is described in Tropos through associations between agents, their goals and the available plans. In addition, the dependencies between the agents are described. MaSE describes the autonomy by *tasks*, which an agent is capable to execute on its own responsibility. Prometheus supports a similar concept, *functionalities* are at the agent's disposal to achieve goals. Therefore, autonomy is clearly expressed by all methodologies. The expressiveness of pro–activity is closely related to the BDI concepts. Tropos and Prometheus are taking advantage of their comprehensive support for BDI notions. The distribution and mobility of agents is only displayed by MaSE, matching Jadex. Both other methodologies just model the acquaintance and communicational relationships between agents.

In [2] the development phases of Tropos are shown in relation to other methodologies by comparing the coverage of different development phases. Figure 3 displays the coverage of the different workflows from the UP in a similar way. To guide grading other prominent approaches are included (Gaia [38] and MESSAGE/UML [5] along with the AUML–Notation[6]). MaSE supports the development from requirements analysis to implementation. In Tropos the analysis of the requirements is more comprehensive. The so called early requirements of the system are modeled according to the i* framework by Yu [39]. Prometheus is also taking testing into account [27]. Guidance in management of an agent oriented software project is fairly small. For the most part, merely heuristics are given.

[6] http://www.auml.org

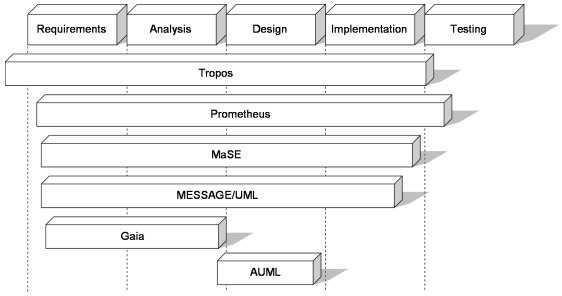

Fig. 3. Coverage of the different workflows from the UP (according to [2])

In examined methodologies the proposed progression in development is clearly described and easy to comprehend. They all propose an iterative approach and develop top–down, from the analysis of the requirements to the identification and description of the single agents.

4.3 Examining the Notation and Tools – Modeling a Case Study

Beginners are guided in the usage of Jadex with a tutorial [1]. In small examples the implementation of agent–oriented concepts is presented. We modeled the last and most complex of these examples that combines the core concepts to a small and therefore easy to comprehend multi agent system. Aim of this system is to translate sentences by forwarding requests for the translation of single words to a dedicated agent, which has access to a dictionary. This translation service is registered at a Directory Facilitator (see [1] for details).

The made experiences in the usage lead to an impression on the expressiveness of notations and usability of the methodology in itself (including the tools). Another promising approach to gain more experiences is modeling a *Challenge Exemplar* as proposed in [40], which leads to a sounder understanding of the strengths and weaknesses of the different methodologies.

MaSE Usage of this methodology showed its focus on modeling the communications. Also flow of control inside a plan is very obviously displayed. The *pragmatics* are dominated by the CASE–Tool *agentTool* 2.0.[7] It is freely available and comfortable to use. We used it merely as a drawing tool, because connectivity is only given for agent platforms which differ fundamentally from Jadex. Methodology and tool are well documented in conference proceedings. According to [8] this methodology has been most widely used in university projects, no industrial use is known.

[7] http://www.cis.ksu.edu/ sdeloach/ai/agentool.htm

Tropos Examining the early requirements has not been suitable for our small case study. These are more concentrated on the situation in which the system to develop is needed. Process and models lead the developers to understand the agent based application as an organization of depending individuals. This is especially valuable for inexperienced users to get used to this new paradigm. While lacking a specialized CASE-Tool, conventional support for UML is suitable during design. For the earlier phases of development, specialized tools supporting the notation from i* can be used.[8] There is no known connectivity to agent platforms from these tools. Documentation is nearly exclusively available as conference submissions, but [2] gives a comprehensive description. Tropos has been used in a few projects (according to [8]).

Prometheus On the design level Prometheus describes implementation related details using descriptors. Central elements are the System and Agent Overview Diagrams. They give a intuitive overview of the system and the agents. This unusual and exceptional not UML based notation is supported by the *Jack Development Environment*, an integrated development environment for the commercial JACK agent platform. Code is directly generated, which means best connectivity for this platform. In addition, the *Prometheus Design Tool*[9] (PDT) is freely available. It is a drawing and documentation tool (no connectivity), which is easy to use and very helpful for the use of the notation. The diagrams give a general impression, but the associated descriptors hold the relevant information. Being able to navigate these descriptors by the visual representations is most valuable and allows to get a quick impression on the single elements. Besides conference contributions there are also useful tutorial notes available [23]. Like in MaSE, this methodology has been used in a number of university projects [8].

Result of Evaluation All three evaluated methodologies are basically capable to support the development of applications using Jadex. A big disadvantage of MaSE is that it does not use BDI concepts throughout the whole development cycle. Prometheus is unique in its detailed description of the individual components forming the agent system and the freely available CASE–Tool. In addition, the used criteria are matching to the greatest extent, the process is nearly as extensive as in Tropos and the modeling language is slightly more comprehensive. Therefore, it is concluded to propose the use of Prometheus for development with Jadex. This leads to considerations how to include Prometheus in development efforts. Since Jadex is a fairly new development there is currently no connectivity between Prometheus and Jadex. Therefore, we evaluated the possible exchange of detailed design information between the above mentioned PDT and the Jadex platform. Agents in Jadex are defined by so called *Agent Definition Files* (ADF). These are XML descriptions of their properties, and a set of Java classes (referenced in the ADF) to implement the desired behavior. In [34] a prototype has

[8] Overview of available tools at: http://www.troposproject.org/
[9] http://www.cs.rmit.edu.au/agents/pdt/

been developed to transform PDT project files into a set of ADFs and vice versa. The match is not comprehensive enough to allow automatic transformation, the developers need to add a number of implementation dependent details to get fully functional ADFs. Since the match between methodology and platform has been examined areas of further improvement to allow transfer of detailed design information between modeling and platform (connectivity) are indicated by the evaluation itself. The discovered mismatches identify, which improvements of agent oriented concepts are needed (in methodology or platform) to cover a common expressiveness. As a partial result of the described evaluation we adopted the Prometheus methodology for a larger research project, which proposes an agent oriented approach to the problem domain of hospital logistics [25].

5 Conclusions

In this paper we presented a flexible framework for evaluation of agent–oriented methodologies that takes platform specific criteria into account. This framework is based on the observation that available agent platforms imply different concepts of agency in different peculiarities. Therefore, the match between methodologies and platforms is examined. A methodology is well suited for a platform if the properties of a methodology match the properties of the platform used for development. The framework stresses that there are both platform dependent and independent criteria to evaluate. The dependent ones need to be derived from the proposed list of abstract concepts with respect to the candidates before they can be examined. Evaluation has to take the nature of criteria into account. Some are suitable for a simple feature–analysis others are more subtle and subjective, their assessment is therefore a difficult task in itself. Case studies, formal experiments and surveys are appropriate for their consideration.
Considering the above described match makes it possible to evaluate different scenarios. It is possible to compare one methodology to many platforms (1:n), several methodologies for one specific platform (n:1) and n:m scenarios where n methodologies are rated to m platforms. This flexibility fits the needs of most software producing organizations. To interpret the evaluation results correctly it is needed to deliberate about the match vs. quality of proposals. A well suited methodology is not only perfectly matching a target platform, but also supports a wide range of platform independent criteria.
The usage of our framework was illustrated by an example evaluation of a group of well known methodologies for their suitability to support development using the Jadex platform. This example usage also illustrated how to derive platform dependent criteria from the proposed abstract concepts set.
Future improvements to the presented framework may result from an examination whether it is useful to consider *application dependent* criteria. For software producing organizations the problem domain according to a concrete project may have impact on the selection of platforms and methodologies.

References

1. Braubach L. und Pokahr A. *Jadex Tutorial* - Release 0.9, 2003. http://sourceforge.net/projects/jadex
2. Bresciani P., Giorgini P., Giunchiglia F., Mylopoulos J., Perini A. *Troops: An agent-oriented software development methodology*, Technical Report DIT-02-0015, University of Trento, 2002.
3. Brooks R. "Elephants Don't play chess". Robotics and Autonomous Systems, 6:3-15, 1990.
4. Busetta P., Howden N., Rönnquist R. und Hodgson A. "Structuring BDI Agents in Functional Clusters". in N. R. Jennings and Y. Lesperance, Intelligent Agents VI. Springer Verlag, Berlin, 1999.
5. Caire G., Leal F., Chainho P., Evans R., Garijo F., Gomez J., Pavon J., Kearney P., Stark J. and Massonet P. "Agent oriented analysis using message/uml". In Agent-Oriented Software Engineering (AOSE), 2001.
6. Cernuzzi L. und Rossi G. "On the evaluation of agent oriented modeling methods", In Proc. of Agent Oriented Methodology Workshop, Seattle, 2002.
7. Corkill D. D. "Blackboard Systems", AI Expert, 6(9):40-47, September, 1991.
8. Dam K. H. and Winikoff M. "Comparing Agent-Oriented Methodologies", In Proc. of the Fifth Int. Bi-Conference Workshop on Agent-Oriented Information Systems (at AAMAS03), 2003.
9. DeLoach S. A. "Analysis and design using MaSE and agentTool". In Proc. of the 12th MAICS, 2001.
10. Ferber J. und Gutknecht O. "A Meta-Model for the Analysis and Design of Organizations in Multi- Agent Systems", In Proc. of the Third Int. Conf. on Multi-Agent Systems (ICMAS98) Paris, France, 1998.
11. Foundation for Intelligent Physical Agents. *FIPA Abstract Architecture Specification*, SC00001L, 2002. http://www.fipa.org/specs/fipa00001/
12. Foundation for Intelligent Physical Agents. *FIPA ACL Message Structure Specification*, SC00061G, 2002. http://www.fipa.org/specs/fipa00061/
13. Georgeff M. and Lansky A. "Reactive Reasoning and Planing: An Experiments With a Mobile Robot", in Proc. of the 1987 National Conference on Artificial Intelligence (AAAI 87), 1987.
14. Henderson–Sellers B. and Gorton I. "Agent-based Software Development Methodologies", White Paper, Summary of Workshop at the OOPSLA 2002, 2003. http://www.open.org.au/Conferences/oopsla2002/index.html
15. Hodgson A., Rönnquist R., Busetta P. *Specification of Coordinated Agent Behavior (The SimpleTeam Approach)*, Technical Report 99-05, Agent Oriented Software Pty. Ltd., 1999.
16. Iglesias C.A., Garijo M. und González J.C. "A Survey of Agent-Oriented Methodologies". In Intelligent Agents V – Proc. of the Fifth Int. Workshop on Agent Theories, Architectures, and Languages (ATAL-98), 1999.
17. Jacobson I., Booch G., Rumbaugh J. *The Unified Software Development Process*. Object Technology Series. Addison Wesley, 1999.
18. Jennings N.R. "On Agent–Based Software Engineering", Artificial Intelligence, 117(2), 2000:277.
19. Jennings N.R. und Wooldridge M. *Agent-Oriented Software Engineering*, Handbook of Agent Technology AAAI/MIT Press, 2000.
20. Kitchenham B. *DESMET: A method for evaluating Software Engineering methods and tools*, Technical Report TR96-09, ISSN:1353-7776, 1996.

21. Luck M., McBurney P. und Preist C. *Agent Technology: Enabling Next Generation Computing: A roadmap for agent–based computing*. AgentLink report, ISBN 0854 327886, 2003. http://www.agentlink.org/roadmap/index.html
22. O'Malley S. A. and DeLoach S. A. "Determining When to Use an Agent-Oriented Software Engineering Paradigm", In Proc. of the AOSE-2001, 2001.
23. Padgham L. "Design of Multi Agent Systems", Tutorial at Net.ObjectDays, October 7-10, 2002, Erfurt, Germany, 2002.
24. Padgham L. und Winikoff M. "Prometheus: A Pragmatic Methodology for Engineering Intelligent Agents", in Proc. of the workshop on Agent-oriented methodologies at OOPSLA 2002.
25. Paulussen T. O., Zöller A., Heinzl A., Pokahr A., Braubach L., Lamersdorf W.: "Dynamic Patient Scheduling in Hospitals" in: Agent Technology in Business Applications (ATeBA-04), 2004.
26. Pokahr A., Braubach L. and Lamersdorf W. *Jadex: Implementing a BDI-Infrastructure for JADE Agents*, EXP – in search of innovation, 3(3):76-85, 2003. http://sourceforge.net/projects/jadex
27. Poutakidis D., Padgham L., Winikoff M.: "Debugging multi-agent systems using design artifacts: The case of interaction protocols". In Proc. of the First Int. Joint Conf. on Autonomous Agents and Multi Agent Systems (AAMAS'02), 2002.
28. Rao A. und Georgeff M. "BDI-agents: from theory to practice". In Proc. of the First Intl. Conf. on Multiagent Systems, 1995.
29. Rational Software White Paper. *Rational Unified Process: Best Practices for Software Development Teams*, 2001. http://www.rational.com/products/whitepapers/100420.jsp
30. Rumbaugh J., Jacobson, I. und Booch, G. *The Unified Modeling Language Reference Manual*, Addison-Wesley, 1999.
31. Rumbaugh J., Blaha M., Premerlani W., Eddy F. und Lorensen W. *Object–Oriented Modeling and Design*, Prentice–Hall, 1991.
32. Shehory O. and Sturm A. "Evaluation of modeling techniques for agent-based systems". In Proc. of the 5th Int. Conf. on Autonomous Agents, ACM Press, 2001.
33. Sturm A. and Shehory O. "A Framework for Evaluating Agent-Oriented Methodologies", Workshop on Agent-Oriented Information System (AOIS), Melbourne, Australia, 2003.
34. Sudeikat J. "Betrachtung und Auswahl der Methoden zur Entwicklung von Agentensystemen", diploma thesis, in German, HAW Hamburg, 2004.
35. Tambe M. "Agent Architectures for Flexible, Practical Teamwork". In Proc. of the Nat. Conf. on Artificial Intelligence, AAAI, 1997.
36. Tveit A. "A survey of Agent-Oriented Software Engineering". In: NTNU Computer Science Graduate Student Conference, 2001.
37. Wood B., Pethia R., Gold L.R. and Firth R. *A guide to the assessment of software development methods*, Technical Report 88-TR-8, Software Engineering Institute, Carnegie- Mellon University, 1988.
38. Wooldridge M. J., Jennings N. R. und Kinny D. "The Gaia methodology for agent-oriented analysis and design". Autonomous Agents and Multi-Agent Systems, 3(3):285–312 , 2000.
39. Yu, E. "Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering". In Proc. of the 3rd IEEE Int. Symp. on Requirements Engineering, 1997.
40. Yu E. and Cysneiros L. M. "Agent-Oriented Methodologies - Towards A Challenge Exemplar". CEUR Workshop Proceedings, 2002.