A Design Complexity Evaluation Framework for Agent-Based System Engineering Methodologies

Anthony Karageorgos and Nikolay Mehandjiev

Department of Computation, UMIST, Manchester M60 1QD, UK
{karageorgos, mehandjiev}@acm.org

Abstract. Complexity in software design refers to the difficulty in understanding and manipulating the set of concepts, models and techniques involved in the design process. Agents are sophisticated software artefacts, associated with a large number of features and therefore Agent-Based System (ABS) engineering methodologies involve considerable design complexity. This paper proposes a framework to evaluate ABS engineering methodologies against a number of criteria related to design complexity. The framework is applied to a number of representative ABS engineering methodologies. The strengths and weaknesses of each methodology with respect to the framework aspects are discussed within the context of a case study involving a virtual enterprise combining manufacturing and logistics services. The evaluation results are used to motivate and guide further work in the area.

1 Introduction

Agent-Based Systems (ABSs) can currently be designed based on ad-hoc methodologies, formal methodologies or informal but structured methodologies. In addition, design can be done either statically, before the ABS is deployed, or dynamically on run-time. All existing methodologies have certain weaknesses and involve considerable difficulty in understanding and manipulating the concepts and models needed for the detailed ABS design. This is referred to as *design complexity*.

The term complexity has been given many definitions in the literature and the majority of them are based on the Oxford English dictionary definition, referring to “difficulty in understanding”. Software engineering complexity relates to how difficult it is to implement a particular computer system [17]. It is considered that high software complexity results in low software quality [10]. In this work, the focus is on ABS engineering complexity and in particular on that related to ABS design.

The sophisticated structure and properties of software agents increase the complexity inherent in ABS design. For example, designing agents to operate in dynamic and open environments and carry out non-trivial tasks that require maximisation of some utility payoff function involves high design complexity [33].

Decreasing software complexity results in reduced time and cost for development and maintenance, fewer functional errors and increased reusability. Therefore, software metrics researchers often try to predict software qualities based on complexity metrics [17]. Furthermore, certain factors are associated with lower complexity. For example, reusing design knowledge reduces design complexity allowing designers to work with concepts of larger granularity at higher abstraction levels [1].

Traditional software engineering methodologies have proven unsuitable for engineering ABSs, and this has spawned new methodologies specifically targeting ABSs. These new methodologies involve different degrees of design complexity because of differences in modelling concepts and techniques used.. For example, the technique of semi-automating the design process results in lower design complexity [14]. Work on reducing ABS design complexity would therefore benefit from a systematic assessment of ABS engineering methodologies with respect to design complexity, which can lead to identifying issues that would need further improvement. To this end, a framework for evaluating design complexity ABS engineering methodologies with respect to is proposed in this paper.

There are only a few attempts to systematically evaluate ABS engineering methodologies. Evaluations are typically done in the context of a case study in order to identify issues that would justify extending a particular methodology as is the case in [13]. However, such evaluations concentrate only on a small number of issues of interest. In other cases, a systematic framework is proposed, which typically focuses on specific parts of the methodologies, for example the expressiveness [7], or trying to provide a high-level overall evaluation of the methodologies as is the case in [8]. Furthermore, a detailed evaluation framework for comparing ABS engineering methodologies, accompanied with references to existing relevant work, is proposed in [26]. However, none of the existing works focuses on assessing the difficulty involved in designing an agent-based system using a particular methodology, which is the focus of this paper.

To demonstrate the evaluation framework proposed in this paper, six representative ABS engineering methodologies were evaluated: RAPPID [23], DESIRE [4], Gaia [34], MESSAGE/UML [5], Tropos [6], the Zeus methodology [21] and KARMA [27]. The evaluation results reveal that the majority of the methodologies examined involve high design complexity as they do not consider organisational settings and collective behaviour as first class design constructs, they do not provide systematic support for design heuristics and non-functional aspects and they do not allow work at a high abstraction level and part of the design process to be carried out automatically. Therefore, further work is required in this direction.

The contents of the paper are as follows. The proposed framework is described in Section 2. Section 3 discusses the results of applying the framework to evaluate a number of representative ABS engineering methodologies. Some issues concerning further research are highlighted in Section 4. Finally, Section 5 concludes the paper.

2 An Evaluation Framework for ABS Design Complexity

The proposed framework was inspired by attempts to discuss the issues involved in ABS design in a systematic manner [11] and it is based on similar work concerning evaluation of object-oriented software engineering methodologies [29], comparison of ABS toolkits [24] and measurement of software complexity [10].

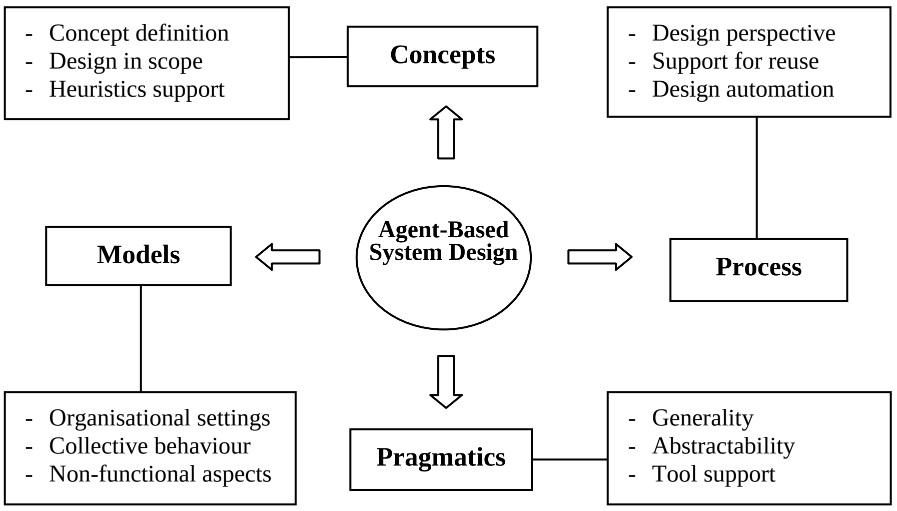

Fig. 1. A framework for assessing the design complexity of ABS engineering methodologies

The framework examines ABS engineering methodologies from four different views, Con*cepts*, *Models*, *Process* and *Pragmatics,* which are summarised in Fig. 1. Each view represents a set of conceptually linked aspects and examines ABS engineering methodologies from a different perspective. For example, the implementation language and the use of standard notations are both related to implementation and hence they should be associated with an implementation-related view.

When assessing an ABS engineering methodology using the proposed framework, a ranking scheme for each aspect is applied. The ranking is based on subjective, qualitative values, for example, low, medium, high. The possible ranking values are discussed together with the different aspects of the framework below. Where appropriate, examples referring to relevant ABS engineering methodologies are provided.

Concepts

The concepts view concentrates on which modelling concepts are used in each methodology to represent the ABS behaviour. It includes the following aspects:

1. *Concept Definition:* This aspect refers to restrictive premises concerning the agent architecture and the type[1] of agents that can be produced with the methodology. Based on this criterion, an ABS engineering methodology can be characterised as *open, bounded* or *limited (highly bounded).* A methodology is open if it does not consider a particular agent architecture and does not produce specific agent types, such as Gaia [34]. An example of a methodology bounded to a particular agent architecture is Tropos [6], which assumes only BDI agents. Finally, a methodology limited to specific agent types is RAPPID [23], which considers

[1] An agent type is a class of agents with similar capabilities and purpose.

only *Component Agents* that represent humans and *Characteristic Agents* that represent parts of a product design system. An open methodology is preferable as it can directly produce different agent types which are most appropriate for different application domains and allows implementation using the programming language or agent toolkit of choice. This results in lower design complexity.

2. *Design in Scope:* This aspect refers to whether a methodology includes specific steps and guidelines for the design phase of the ABS engineering lifecycle and can be *true* or *false*. For example, MESSAGE/UML [5] covers only the analysis phase while Tropos [6] covers analysis, design and also part of the implementation. Explicitly supporting the design phase results to lower design complexity.

3. *Heuristics support:* This aspect refers to the explicit support for applying heuristic guidelines and tips when designing the ABS and can be *true* or *false*. Explicit heuristics support involves providing formal techniques that can be used to ensure application of the design heuristics. For example, in KARMA [27] heuristics can be specified as constraints in the STEAM specification language. In contrast, in RAPPID [23] there is no rigorous way for ensuring that design heuristics have been applied. Formal heuristics support results in lower design complexity.

Models

The *Models* view refers to the models that are used to represent different parts of the ABS or issues of particular interest and the techniques that are used to create and manipulate those models. The *Models* view includes the following aspects of interest:

1. *Organisational settings:* This framework aspect concerns whether organisational settings are considered as first-class design constructs and can be *true* or *false*. For example, in the ABS engineering methodology associated with the Zeus agent toolkit [21] organisational settings are represented by explicit role models in contrast to DESIRE [4] where they are implied by the agent behaviour. Organisational settings should be considered as first class design constructs [22], [35], enabling work in higher abstraction levels and thus lowering the design complexity.

2. *Collective Behaviours:* This aspect refers to whether an approach includes appropriate first-class modelling constructs to represent collective agent behaviour and it can be *true* or *false*. Representation of collective behaviour can be implicit via the individual agent behaviour such as in RAPPID [23], or explicit; for example, in the Zeus methodology it is represented by role models [21]. Collective behaviours should be considered as first class design constructs enabling reasoning at a high abstraction level [15] and hence resulting to lower design complexity.

3. *Non-functional aspects:* This aspect refers to whether non-functional aspects are explicitly considered in the methodology and can be *true* or *false*. Non-functional aspects can be explicitly represented by appropriate modelling constructs, such as in Tropos [6], or they can be implicitly modelled within individual agent behaviour such as in Gaia [34]. Explicitly modelling non-functional aspects enables work at a higher abstraction level and results in lower design complexity.

Process

The process view concentrates on the steps that are executed to construct the models discussed in the *Models* view and on techniques that support and assess those steps. In particular, this view is concerned with the following aspects:

1. *Design Perspective:* This aspect refers to the perspective from which each methodology views the ABS design. The perspective can be *top-down* or *bottom- up* or *both* (top-down and bottom-up) depending on how the design of the ABS progresses. In the top-down perspective, the design models are constructed by refining high-level models of the agent organisation, such as in Gaia [34]. In the bottom-up perspective, design models are progressively composed from existing finer-grain models thus enabling reuse [15]. Supporting both perspectives, as in MESSAGE/UML [5], can help to reduce design complexity.

2. *Support for Reuse:* This aspect refers to whether the methodology supports using previous knowledge in designing an ABS and can be *true* or *false*. Support for reuse involves modelling constructs, techniques and guidelines for the identification, representation, testing and application of reusable knowledge. For example, in the Zeus toolkit methodology [21] there are guidelines for creating, storing and reusing negotiation strategies when specifying agent interactions, whilst in RAPPID [23] there are not such facilities. Support for reuse is a fundamental step towards achieving lower design complexity [1].

3. *Design Automation:* This aspect refers to whether there are formal underpinnings in the specification models of the methodology enabling automation of the design process to a certain extent. Some process steps should definitely be carried out based on the judgement of the human designers, for example the selection of roles in the analysis phase in Gaia [34]. However, other steps could be automated and carried out by a software tool, for example based on formal model transformations [25]. The degree to which the process steps are automated can be characterised as *true* or *false*. For example, the DESIRE [4] design process can be automated, as many steps are formally defined using mathematical techniques, in contrast to RAPPID [23] where there are no formal underpinnings. Automating the design process results in lower design complexity and reduces development effort and errors [1].

Pragmatics

This view focuses on the pragmatics of each ABS engineering methodology. In other words, this view refers to how practical the methodology is for the design of real-world agent systems. It is concerned with the following aspects:

1. *Generality:* The generality of a methodology refers to the existence of restrictive premises concerning the environment and the application domain that affect the applicability of the methodology and can be characterised as *high, medium* or *low*. High generality means that the methodology can be applied without any significant restrictions, such as Tropos [6]. The generality is medium when there are considerable restrictions but the applicability of the method is still wide. For example, Gaia [34] assumes closed ABSs and small numbers of cooperating agents.

In contrast, RAPPID [23] is limited since it can only be applied to design ABSs that will be used to support industrial product design and, therefore, its generality is low. High generality results to lower design complexity since it is easier to apply the methodology in various application domains.

2. *Abstractability:* This aspect refers to whether there is support to enable work at different levels of abstraction which is one of the main factors affecting design complexity [1] and it can be *true* or *false*. For example, role-based methodologies, such as [15], support abstractability since agent behaviour can be specified at both the level of roles and at the level of role characteristics. In contrast, in Tropos [6] this is done only at the agent level and hence Tropos does not support abstractability.

3. *Tool support:* This aspect is concerned with whether there are tools supporting the realisation of the methodology. For example, the Zeus methodology [21] is supported by the Zeus agent building toolkit, which assists the users in designing ABSs. On the other hand, there is no tool support for the Gaia approach [34] and the engineer is responsible for manually creating all the relevant models. The tool support of an approach can be characterised as *true* or *false.* It is preferable for an approach to be supported by CASE tools since this reduces development effort and development errors [17] and automates repetitive tasks [20] increasing the usability of the methodology and resulting to lower design complexity.

It must be noted that some aspects are interrelated. For example, low or limited concept definition is likely to be combined with low or medium generality, as is the case in RAPPID [23]. However, this is not always the case, For example, Tropos [6] is bounded to only BDI agents and it is still applicable in many application domains.

3 A Motivating Case Study Example

To better illustrate the different aspects of the framework we consider a case study involving providing support for cross-organisational business process management in virtual enterprises using agent technology (Fig. 2). An example of such an effort is the MABE research project [28]. The requirements imposed by this case study are used to illustrate the strengths and weaknesses of the six ABS engineering methodologies evaluated in Section 4.

For the needs of this example we can assume a Virtual Enterprise (VE) consisting of manufacturers, suppliers and logistic service providers spanning all over the world. Each VE partner operates its business in different local economic, cultural and political conditions, uses various types of legacy systems and software technologies and has different business interests. However, after becoming members of the VE all partners comply with the VE business rules and operational regulations.

The software enabling such interoperation is based on agent technology. All interacting parties including the VE partners plus various VE establishment and administration bodies are associated with appropriate agent components capable of carrying tasks in a distributed and autonomous manner. The agent components are backed by a software infrastructure complementing the agent functionality, for example providing user interfaces to agents, linking agents with external web-services and other information sources and assisting agents in knowledge discovery by implementing data mining and knowledge management algorithms. The only requirement for each VE partner is to install agent software capable of interoperating with that of the existing VE partners and the main VE administration body. Furthermore, the VE partner software will have to comply with the functional specifications designated by the VE establishment body and implement the minimum required functionality.

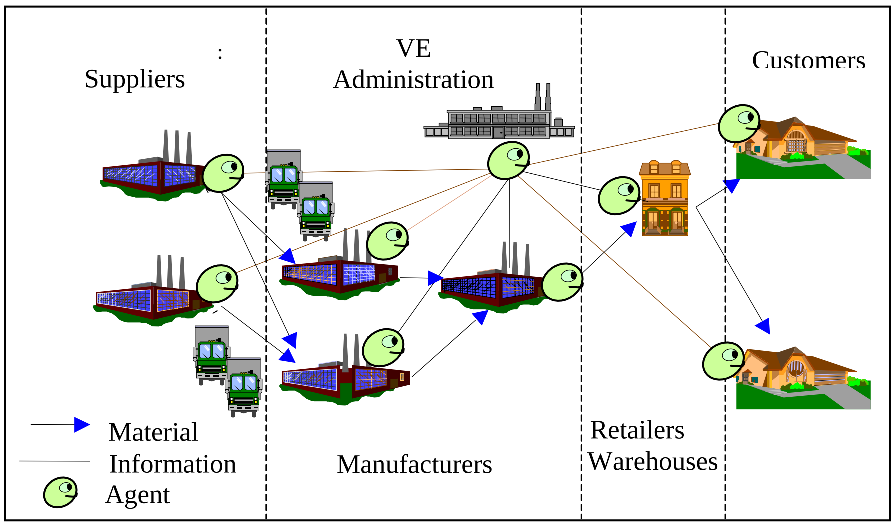

Fig. 2. Agent support for Manufacturing and Logistics Virtual Enterprises

It can be assumed that the agent software is built according to FIPA standards, for example agents communicate using a FIPA compliant communication language. Furthermore, discovery of agent services could be done using a central agent service registry providing white and yellow page information to the eligible agents as is currently done in the Agentcities project [31]. The VE administration body exercises control on the operation of the VE and interacts with all the agents in the VE.

The underlying philosophy in this case study is that the agent system design will be done once by the VE establishment body and the relevant parts will become available to each new VE partner after it has been accepted to join the VE. The new VE partner will have to either develop the necessary software according to the given specifications or purchase it from other partners or software vendors.

In designing the agent-based system certain authority relationships need to be ensured between the main VE agents and the VE member agents. For example, main VE agents should be able to exercise control on VE member agents with respect to providing information about transactions, pricing quotes and service details. Furthermore, several widely used patterns of interacting agent behaviour such as the mediator pattern [18] need to be applied for security and privacy reasons. For similar reasons, a number of heuristics have to be followed in the design, for example, all financial transactions for each VE member should be carried out by an authorised agent, which is recognised by the main VE financial controller agent. Furthermore, since the resulting agent system is large and complicated, a design which allows black-box re-use of different components is to be preferred. Finally, this complex agent system design exercise would be significantly facilitated if a software tool could carry out certain routine but tiresome and error-prone design steps, such as automatically combining known design patterns and imposing constraints on the design product.

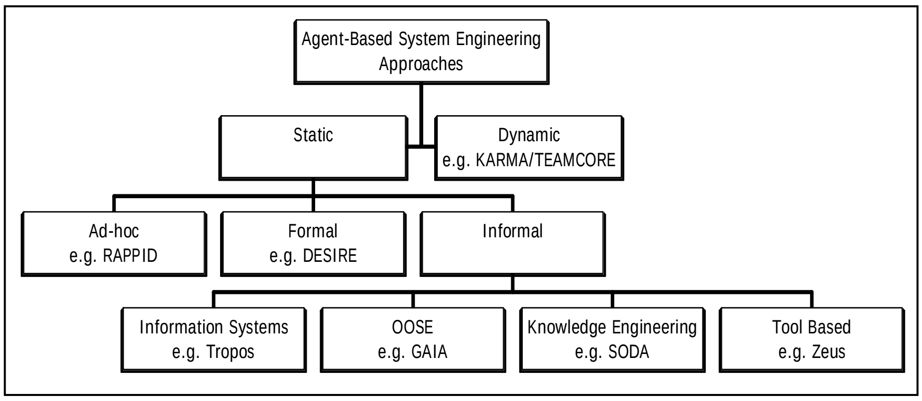
Agent-Based System Engineering Approaches
Static
Dynamic e.g. KARMA/TEAMCORE
Ad-hoc e.g. RAPPID
Formal e.g. DESIRE
Informal
Information Systems e.g. Tropos
OOSE e.g. GAIA
Knowledge Engineering e.g. SODA
Tool Based e.g. Zeus

Fig. 3. Classification of Agent-Based System Engineering Methodologies

4 Comparative Evaluation of ABS Engineering Methodologies

ABS engineering methodologies can be classified as ad-hoc, formal, informal and structured, and dynamic (see Fig. 3). Ad-hoc methodologies involve designing an ABS in an application domain specific manner while formal approaches are based on the use of formal methods. Informal and structured methodologies originate from knowledge engineering and software engineering and are predominantly extensions of object-oriented analysis and design methodologies. Finally, dynamic methodologies involve defining the structure of an ABS and the behaviour of the individual agents dynamically on run-time. All classes have advantages and disadvantages with informal and structured methodologies being regarded as more practical for numerous real-world applications.

A representative methodology of each class (RAPPID [23], DESIRE [4], Gaia [34], MESSAGE /UML [5], Tropos [6], Zeus methodology [21] and KARMA [27]) has been evaluated using the evaluation framework described in Section 2. A summary of the results is presented in Table 1. A more detailed discussion of this classification scheme and a review of the above ABS engineering methodologies can be found in [14].

Regarding the *Concepts* perspective, about half of the ABS engineering methodologies examined (DESIRE, Tropos and Zeus methodology) are bounded to specific agent architecture. RAPPID is the only one limited to specific agent types as well. However, based on the case study described in Section 3, it is clear that a methodology not bounded to specific agent architecture is needed. For example, some VE partners may prefer to develop the agent software using tools they are familiar with or they may want to market their software to other potential VE members. In such a case, proprietary development tools or publicly available tools which are released under a suitable licence (such as JADE [30] that comes under LGPL[2]) will need to be used.

The majority of the methodologies examined (DESIRE, Gaia, Tropos, Zeus methodology and KARMA) consider design as an explicit step in the ABS engineering lifecycle. This is an important requirement of the VE case study since otherwise it would be difficult to design such a large business system without making errors. However, only KARMA provides formal support for heuristics in the design of the ABS and as mentioned in the case study scenario, being able to apply design heuristics in a systematic manner is needed to facilitate such a complex design task. Clearly, this is a general deficiency of current ABS engineering methodologies.

Concepts	**RAPPID**	**DESIRE**	**Gaia**	**MESSAGE**	**Tropos**	**Zeus**	**KARMA**
Concept definition	≤≥	<>	><	><	<>	<>	><
Design in scope	–	√	√	–	√	√	√
Heuristics support	–	–	–	–	–	–	√
Models							
Organisational settings	–	–	–	–	–	√	√
Collective behaviour	–	–	–	–	–	√	√
Non-functional aspects	–	–	–	–	√	–	–
Process							
Design perspective	↓	↓	↓	↕	↓	↑	↓
Support for reuse	–	√	–	–	–	√	–
Design automation	–	–	–	–	–	–	√
Pragmatics							
Generality	○	∅	∅	⊗	⊗	∅	⊗
Abstractability	–	√	–	–	–	–	√
Tool support	–	√	–	√	–	√	√

Legend

○ - low ∅ - medium ⊗ - high	≤≥ - limited <> - bounded >< - open	↑ - bottom-up ↓ - top-down ↕ - both	√ - yes – - no

Table 1. Design complexity evaluation of ABS engineering methodologies

[2] LGPL stands for Lesser General Public Licence and it allows extensions to the original software to be released under any, even commercial, licence.

As far as it concerns the *Models* perspective, only the Zeus methodology and KARMA explicitly model organisational settings. Representing collective behaviours as first class design constructs is also not supported in most of the examined methodologies. The only exceptions are Zeus where collective behaviours can be represented by role models and KARMA where collective behaviours are modelled by appropriate team plans. The lack of support for non-functional aspects is even more pronounced. Indeed, only Tropos considers non-functional aspects in the design of ABSs. As discussed in the previous section, all three aspects are needed to efficiently design the ABS for the case study considered.

In the *Process* perspective, only MESSAGE/UML allows working in both top-down and bottom-up fashion (but the current version of MESSAGE/UML supports only the analysis phase of the ABS engineering lifecycle and hence it is not practically useful). The Zeus methodology supports bottom up design, the rest of the approaches are all supporting top-down design. In the case study considered, both top-down and bottom-up design would be needed. The VE administrative structure would be easier to be modelled in a top-down manner. On the other hand, agent behaviours supporting VE partners would be more practical to be synthesised bottom-up to better reflect the localised dependencies and requirements.

Furthermore, only two approaches explicitly provide support for reuse, DESIRE and the Zeus methodology. DESIRE includes guidelines about how the agent system designer can reuse generic task components in the design of the ABS and the Zeus methodology includes guidelines about how to reuse generic behaviours represented by role models and generic agent characteristics — for example negotiation strategies. Support for reuse is mandatory in the case study considered, as large parts of the required agent software functionality, such as contracting and negotiation mechanisms, have already broadly implemented and tested.

There is also significant lack of support for automatic design of ABSs. Only KARMA supports automatic selection of the agents that will participate in the agent organisation based on team plans specified by the designer. This is also a mandatory requirement in the case study scenario due to the size of the agent application that needs to be designed.

Regarding the *Pragmatics* perspective, approximately half of the approaches (MESSAGE, Tropos and KARMA) are general, targeting a broad range of application domains. The rest are restricted as follows: Gaia assumes closed ABSs consisting of small numbers of static, cooperating agents. The Zeus methodology has restrictions regarding the environments where the agents produced can operate. For example, Zeus agents cannot be mobile and they require a large amount of physical RAM memory to execute. DESIRE is also specific to applications requiring static agents whose behaviour can be described by a task-based hierarchy. RAPPID is the most specific approach since it targets a specific application domain; that of supporting industrial product design. The case study example considered clearly highlights the need for a general methodology. The VE is an open system where partners can dynamically join and deregister and therefore it must be supported by an open agent system. Mobility can also be useful, for example for realising ant-based coordination algorithms. Finally, the case study considered concerns the domain of manufacturing and logistics service provision and specific methodologies concerning other domains, such as RAPPID, are not applicable.

DESIRE formally supports specifying interactions among task components at different levels of abstraction, which reduces design complexity. This is the case for KARMA, which is based on the STEAM formal specification language. STEAM makes possible for the designer to work at different levels of abstraction with appropriate rigour. The rest of the tools do not support abstractability and the designer has to manually consider all the design details. This is something that would not suit the case study considered. For example, common functionality such as interactions with the VE administration and transaction handling and logging would need to be specified in detail for each agent type.

Finally, four of the methodologies examined (DESIRE, MESSAGE, Zeus and KARMA) are associated with tools that assist the designers in applying them. The assistance provided includes graphic user interfaces behaviour allocation mechanisms as well as automatic design product construction, verification and generation of the source code. Clearly, these are necessary features for the methodology selected to be applied to the design of the required in the case study considered in this paper.

5 Implications for Further Research

The above analysis has demonstrated that none of the ABS engineering methodologies examined covers all aspects of design support included in the evaluation framework introduced in Section 2. An effective approach to ABS design should therefore cover a number of outstanding issues, which are described in more detail below.

5.1 Support for Design Heuristics

Existing ABS engineering methodologies do not provide systematic and rigorous models for considering heuristics in the design of the ABS. In methodologies having formal underpinnings, such as DESIRE [4], design heuristics can be taken into account in a rigorous manner in the design but there are no guidelines and systematic techniques assisting in this task. The designer needs to manually incorporate the heuristic rules in the formal ABS specifications.

Some methodologies support informal ABS design heuristics. For example in the Zeus methodology [21] the *sphere of responsibility* and *point of interaction* heuristics are provided. The former requires the designer to partition the application resources to areas of control and represent each area with a software agent. The latter refers to representing each resource in the application domain with an agent. However, those informal heuristics cannot be easily applied to the design of large ABSs. Furthermore, it is difficult for the designer to predict the effect on design decisions when those heuristics contradict with other requirements such as non-functional requirements. Hence, new ways to support heuristics in ABS are required.

5.2 Organisational Settings

Some ABS engineering methodologies explicitly model organisational settings — for example, MAS-CommonKADS [12] and SODA [22] — and there are cases where the agent organisation is designed during a distinct design step, before the agent behaviour is completely specified [3]. However, it has been argued that even when organisational settings are explicitly modelled, the models only represent the organisational relationships between agents without considering social tasks and social laws [36]. Furthermore, organisational settings are not considered as first class design constructs apart from a few exceptions of approaches that use roles [21], [22]. Another problem concerning organisational settings is that existing approaches do not provide rigorous methodologies for combining organisational settings with application functionality. This has to be done intuitively by the designer without any assistance by a software tool.

5.3 Collective Behaviour

A similar problem exists regarding representing collective behaviour. Many authors argue that collective behaviours should be treated as first-class design constructs, namely that they should be able to be instantiated and given identity [2], [15]. However, even where this is issue is addressed, such as in the Zeus methodology [21], there is no rigorous way to reuse collective application functionality and combine it with organisational settings.

5.4 Non-Functional Aspects

An issue of major concern in ABS design is the modelling and consideration of non-functional aspects such as security and performance. To the best of author's knowledge, no ABS engineering approach explicitly considers non-functional aspects in design apart from Tropos [6], In Tropos, the software system is represented as one or more actors, which participate in a strategic dependency model, along with other actors from the system's operational environment. Actors can fulfill certain goals and are progressively identified and inserted in the conceptual models by the agent system engineers. In this way, non functional aspects can be considered if engineers insert appropriate actors with goals that contribute positively to the satisfaction of non-functional requirements in the Tropos conceptual models. However, the Tropos approach to modelling non-functional aspects suffers from two main weaknesses. Firstly, it models non-functional aspects in a way that it cannot be directly reused in other ABS designs. Secondly, quantitative characterisation of non-functional aspects is not possible.

In some cases, non-functional aspects are the basis for criteria for reorganisation in dynamic approaches, as is the case in KARMA [27]. In these instances non-functional aspects are taken into account by adjusting the agent behaviour and the organisation of the ABS at run-time. However, this treatment of non-functional aspects impedes the reuse of non-functional models. It also contributes to significant consumption of resources and may cause system instability.

5.5 Automating the Design Process

In order to reduce development effort and software design errors the design process should be partially automated [16]. This view is also adopted by some ABS engineering methodologies [9], [25], [32] which are considered informal because their analysis models use informal specifications of desired systems. They nevertheless try to provide the formal underpinnings necessary for automatically designing ABSs. The common way of doing that is by progressing from analysis to design by successive formal transformations of the analysis models. The transformations used, however, focus on ensuring that the designed agent components are correctly represented in respect to the analysis models, using object-oriented software engineering concepts and techniques. For example, in [25] formal transformations are used to decide on the number of objects and concurrent threads that should be used to correctly realise the behaviour of each agent component. To the best of author's knowledge, current informal ABS engineering approaches do not provide any automatic support for actually deciding on what behaviour each agent in the ABS should have. This is not the case for dynamic approaches where the design of the agent system is done during reorganisation steps. For example, in KARMA the agent components are automatically selected based on specifications of the agent-based application requirements described in the STEAM modelling framework [27]. However, KARMA assumes that agents already exist in cyberspace, which is not generally the case.

5.6 Working at Different Abstraction Levels

There is a consensus that abstraction in software design reduces design complexity [19]. Although it has the trade-off of reducing software efficiency and performance, it may add to the reliability of the produced software as frequently used components are thoroughly tested and the design process can be automated [1].

As abstraction is a common practice in software design, a number of ABS engineering methodologies allow the designer to work at different levels of abstraction. However, not all of them provide appropriate formal support. For example, MESSAGE/UML allows modelling at levels 0 and level 1 but there is no formal description of the relations between the models of the two levels. As a result, proper use of MESSAGE/UML requires the designers to have a clear understanding and explicitly consider the links between models at levels 0 and 1, which makes the ABS design task more difficult,

The only approaches examined that provide formal support for working at different levels of abstraction are DESIRE [4] and KARMA [27]. However, their support is limited. DESIRE only supports interaction between tasks at different abstraction levels and KARMA supports teamwork at different levels of abstraction in the form of joint intentions. Agent behaviour, however, is characterised with other aspects as well. For example, coordination protocols or negotiation strategies, which the designer should specify at the lowest level of detail in those two approaches. This problem is addressed in the Zeus approach [21]. For example, in the Zeus methodology, the agent system designer can either select a predefined negotiation strategy or specify all negotiation rules in detail. Zeus models agent behaviour at different levels of abstraction based on role modelling. However, this support is informal since the relations among roles have not been given formal semantics.

6 Summary

This paper proposed a framework to assess ABS engineering methodologies with respect to design complexity they involve. Using this framework, a set of representative methodologies has been examined revealing a number of issues that would require further research.

The proposed framework suggests looking into ABS engineering approaches from four views: *Concepts*, *Models*, *Process* and *Pragmatics*. The *Concepts* view refers to the modelling concepts used to model ABSs and it concerns the generality of the concept definition, the existence of specific support for design in the ABS engineering process and the support for design heuristics. The *Models* view refers to modelling of organisational settings and collective behaviour to be used as first class design constructs and to explicit modelling of non-functional aspects. The *Process* view examines the perspective of the design process and whether it can be based on reuse and if it can be automated. The *Pragmatics* view evaluates the applicability of the approach to real-world applications by assessing the generality, the complexity handling and the tool support of the approach.

None of the methodologies examined supports all aspects of the proposed framework. Significant gaps have been uncovered to inform further work on ABS engineering methodologies, where the aim would be to decrease design complexity by providing more comprehensive support for all aspects of the framework. It is the authors' belief that using roles as behavioural modelling constructs and providing appropriate semantics for role relations and role characteristics is the most appropriate path to follow towards achieving this goal.

References

1. Alagar, V.S., Periyasamy, K.: *Specification of Software Systems*. New York: Springer-Verlag, 1998.
2. Andersen, E.P. *Conceptual Modelling of Objects: A Role Modelling Approach*. PhD Thesis. Oslo, Norway: University of Oslo, 1997.
3. Barber, K.S., Liu, T.H., Han, D.C. *Agent-Oriented Design*. Austin, TX, USA: University of Texas at Austin, 1999, http://powerlips.ece.utexas.edu/pubs/techReports/1999/TR99-UT-LIPS-AGENTS-01.pdf.
4. Brazier, F.M.T., Dunin-Keplicz, B., Jennings, N., Treur, J.: *DESIRE*: Modelling Multi-Agent Systems in a Compositional Formal Framework. *International Journal of Cooperative Information Systems, Special Issue on Formal Methods in Cooperative Information Systems: Multi-Agent Systems*, 5, 1 (June 1997), 67-94.
5. Caire, G., Coulier, W., Garijo, F., Gomez, J., Pavon, J., Leal, F., Chainho, P., Kearney, P., Stark, J., Evans, R., Massonet, P. Agent Oriented Analysis Using Message/UML. In Wooldridge, M.J., Weis, G., Ciancarini, P. (eds.), *Agent-Oriented Software Engineering II, Second International Workshop, (AOSE 2001), Montreal, Canada*. Berlin: Springer Verlag, 2002, 151-168.
6. Castro, J., Kolp, M., Mylopoulos, J.: Towards requirements-driven information systems engineering: the Tropos project. *Information Systems*, 27, 6 (September 2002), 365-389.
7. Cernuzzi, L., Rossi, G. On the Evaluation of Agent Oriented Methodologies. *OOPSLA 2002 Workshop on Agent-Oriented Methodologies*. 2002.
8. Dam, K.H., Winikoff, M. Comparing AgentOriented Methodologies. *Fifth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2003)*. Malbourne, Australia, 2003.

9. Depke, R., Heckel, R., Kuster, J.M. Agent-Oriented Modelling with Graph Transformation. In Ciancarini, P., Wooldridge, M. (eds.), *Agent-Oriented Software Engineering I, First International Workshop (AOSE 2000), Limerick, Ireland*. Berlin: Springer-Verlag, 2001, 106-119.
10. Fenton, N., Pfleeger, S.L.: *Software Metrics: A Rigorous and Practical Approach*. Boston, MA, USA: PWS Publishing Co., 1997.
11. Iglesias, C.A., Garrijo, M., Gonzalez, J.C. A Survey of Agent-Oriented Methodologies. In Muller, J., Singh, M.P., Rao, A.S. (eds.), *Proceedings of the 5th International Workshop on Intelligent Agents {V}: Agent Theories, Architectures, and Languages (ATAL-98)*. Heidelberg, Germany: Springer-Verlag, 1999, 317-330.
12. Iglesias, C.A., Garijo, M., Gonzalez, J.C., Velasco, J.R. Analysis and Design of Multiagent Systems using MAS-CommonKADS. In Singh, M.P., Rao, A.S., Wooldridge, M.J. (eds.), *Intelligent Agents IV: Agent Theories, Architectures, and Languages (ATAL '97)*. Berlin, Germany: Springer Verlag, 1998, 313-326.
13. Juan, T., Pearce, A., Sterling, L. ROADMAP: Extending the Gaia Methodology for Complex Open Systems. *Autonomous Agents and Multi-Agent Systems (AAMAS 2002)*. Bologna, Italy: ACM Press, 2002.
14. Karageorgos, A. *Using Role modelling and Synthesis to Reduce Complexity in Agent-Based System Design*. PhD Thesis. Manchester, UK: University of Manchester Institute of Science and Technology, 2003.
15. Kendall, E.A.: Role models - patterns of agent system analysis and design. *BT Technology Journal*, 17, 4 (October 1999), 46-57.
16. Lowry, M.R., McCartney, R.D. (eds.). *Automating Software Design*. Menlo Park, CA: AAAI Press, 1991.
17. MacDonell, S.G.: Determining delivered functional error content based on the complexity of CASE specifications. *New Zealand Journal of Computing*, 5, 1 (July 1994), 57-65.
18. Maturana, F., Norrie, D.H.: Multi-agent mediator architecture for distributed manufacturing. *Journal of Intelligent Manufacturing*, 7, 257-270.
19. Metzger, A., Queins, S. A Reuse- and Prototyping-based Approach for the Specification of Building Automation Systems. In Schuerr, A. (ed.), *OMER-2 Workshop Proceedings*. Munich: University of the Federal Armed Forces, Germany, 2001, 3-9.
20. Ng, K., Kramer, J., Magee, J.: A CASE Tool for Software Architecture Design. *Automated Software Engineering*, 3, 3/4 (1996), 261-284.
21. Nwana, H.S., Ndumu, D.T., Lee, L.C., Collis, J.C.: Zeus: A Toolkit for Building Distributed Multi-Agent Systems. *Applied Artificial Intelligence Journal*, 13, 1 (January 1999), 129 - 185.
22. Omicini, A. SODA : Societies and Infrastructures in the Analysis and Design of Agent-based Systems. In Ciancarini, P., Wooldridge, M.J. (eds.), *Agent-Oriented Software Engineering I, First International Workshop (AOSE 2000), Limerick, Ireland*. Berlin: Springer Verlag, 2001, 185-193.
23. Parunak, V.D., Sauter, J., Fleischer, M., Ward, A.: The RAPPID Project: Symbiosis between Industrial Requirements and MAS Research. *Autonomous Agents and Multi-Agent Systems*, 2, 2 (June 1999), 111-140.
24. Silva, A.R., Romao, A., Deugo, D., Silva, M.M.d.: Towards a Reference Model for Surveying Mobile Agent Systems. *Autonomous Agents and Multi-Agent Systems*, 4, 3 (September 2001), 187-231.
25. Sparkman, C.H., DeLoach, S.A., Self, A.L. Automated Derivation of Complex Agent Architectures from Analysis Specifications. In Wooldridge, M.J., Weis, G., Ciancarini, P. (eds.), *Agent-Oriented Software Engineering II, Second International Workshop (AOSE 2001), Montreal, Canada*. Berlin: Springer Verlag, 2002, 278-296.

26. Sturm, A., Shehory, O. A Framework for Evaluating Agent-Oriented Methodologies. *Fifth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2003)*. Malbourne, Australia, 2003.
27. Tambe, M., Pynadath, D.V., Chauvat, N.: Building Dynamic Agent Organisations in Cyberspace. *IEEE Internet Computing*, 4, 2 (March/April 2000), 65-73.
28. The MABE Consortium. *The MaBE (Mulii-Agent Business Environement) Project*. 2003, http://www.mabe-project.com.
29. The Object Agency Inc. A Comparison of Object-Oriented Development Methodologies. The Object Agency, Inc, (Autumn 1995), http://www.toa.com/pub/mcr.pdf.
30. TILAB - Motorola. *The Jade Agent Building Toolkit*. 2003, http://sharon.cselt.it/projects/jade/.
31. Willmott, S.N., Dale, J., Burg, B., Charlton, C., O'brien, P.: Agentcities: A Worldwide Open Agent Network. *Agentlink News*, 8, 13-15.
32. Wood, M., DeLoach, S.A. An Overview of the Multiagent Systems Engineering Methodology. In Ciancarini, P., Wooldridge, M.J. (eds.), *Agent-Oriented Software Engineering I, First International Workshop (AOSE 2000), Limerick, Ireland*. Berlin: Springer Verlag, 2001, 207-221.
33. Wooldridge, M.: On the Sources of Complexity in Agent Design. *Applied Artificial Intelligence*, 14, 7 (August 2000), 623-644.
34. Wooldridge, M., Jennings, N.R., Kinny, D.: The Gaia methodology for agent-oriented analysis and design. *International Journal of Autonomous Agents and Multi-Agent Systems*, 3, 3 (September 2000), 285-312.
35. Zambonelli, F., Jennings, N.R., Wooldridge, M. Organisational Abstractions for the Analysis and Design of Multi-Agent Systems. In Ciancarini, P., Wooldridge, M.J. (eds.), *Agent-Oriented Software Engineering I, First International Workshop (AOSE 2000), Limerick, Ireland*. Berlin: Springer Verlag, 2001, 235-250.
36. Zambonelli, F., Jennings, N.R., Omicini, A., Wooldridge, M.J. Agent-Oriented Software Engineering for Internet Applications. In Omicini, A., Zambonelli, F., Klusch, M., Tolksdorf, R. (eds.), *Coordination of Internet Agents: Models, Technologies and Applications*. Berlin Heidelberg: Springer-Verlag, 2001, 326-346.