# A Design Complexity Evaluation Framework for Agent-Based System Engineering Methods

Anthony Karageorgos[1], Nikolay Mehandjiev[1] and Simon Thompson[2]

[1] Department of Computation, UMIST, Manchester M60 1QD, UK
{karageorgos, mehandjiev}@acm.org
[2] Intelligent Business Systems, BT Exact Technologies, Ipswich IP5 3RE, UK
simon.2.thompson@bt.com

**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 methods involve considerable design complexity. This paper proposes a framework to evaluate ABS engineering methods against a number of design complexity related criteria. The framework is applied to a number of representative ABS engineering methods and the results are used to motivate and guide further work in the area.

## 1 Introduction

ABSs can currently be designed using a number of ad-hoc methods, formal methods or informal but structured methods. In addition, design can be done either statically, before the ABS is deployed, or dynamically on run-time. All existing methods 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 [14]. It is considered that high software complexity results to low software quality [8]. 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 [25].

Lower software complexity provides advantages such as lower development and maintenance time and cost, less functional errors and increased reusability. Therefore, it is common in software metrics research to try to predict software qualities based on complexity metrics [14]. Furthermore, certain factors associated with lower complex-

ity can be identified. For example, reusing design knowledge reduces design complexity allowing designers to work with concepts of larger granularity at higher abstraction levels [1].

The unsuitability of traditional software engineering methods has spawned new methods specifically targeting ABSs. The new methods utilize a variety of modelling concepts and techniques and involve different degrees of design complexity. For example, semi-automating the design process results to lower design complexity [11]. It is therefore necessary to assess ABS engineering methods with respect to design complexity and to identify issues that would need further improvement. To this end, a framework for evaluating ABS engineering methods with respect to design complexity is proposed in this paper.

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 methods. 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 understand and discuss the issues involved in ABS design in a systematic manner [9] and it is based on similar work concerning evaluation of object-oriented software engineering methods [23], comparison of ABS toolkits [20] and measurement of software complexity [8].

The framework examines ABS engineering methods 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 methods 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 method 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 methods are provided.

**Concepts**

The concepts view concentrates on which modelling concepts are used in each method 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 method. Based on this criterion, an ABS engineering method can be characterised as *open*, *bounded* or *limited (highly bounded)*. A method is open if it does not consider a particular agent architecture and does not produce specific agent types,

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

such as Gaia [26]. An example of a method bounded to a particular agent architecture is Tropos [5], which assumes only BDI agents. Finally, an example of a method limited to specific agent types is RAPPID [19], which considers only *Component Agents* that represent humans and *Characteristic Agents* that represent parts of a product design system. It is preferable for a method to be open as it can directly produce various agent types resulting to lower design complexity.

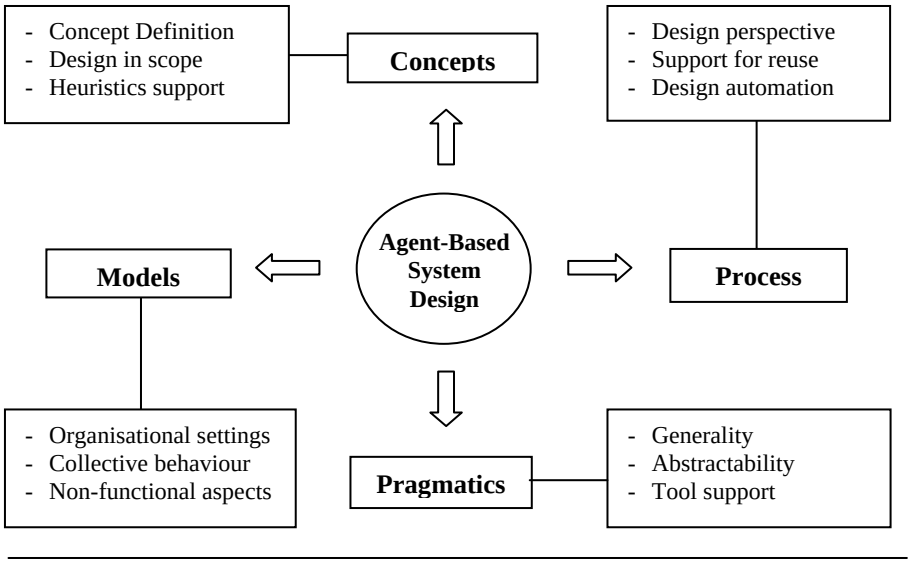



**Fig. 1.** A framework for assessing the design complexity of ABS engineering methods

2. *Design in Scope:* This aspect refers to whether a method includes specific steps and guidelines for the design phase of the ABS engineering lifecycle and can be *true* or *false*. For example, MESSAGE/UML [6] covers only the analysis phase while Tropos [5] 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 formal support for applying heuristic guidelines and tips when designing the ABS and can be *true* or *false*. Formal heuristics support involves providing formal techniques that can be used to ensure application of the design heuristics. For example, in KARMA [22] heuristics can be specified as constraints in the STEAM specification language. In contrast, in RAPPID [19] there is no rigorous way for ensuring that design heuristics have been applied. Formal heuristics support results to 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 Zeus [17] 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 [18], [27], enabling work in higher abstraction levels and resulting to lower design complexity.

2. *Collective Behaviours:* This aspect refers to whether an approach includes appropriate first-class modelling constructs to represent collective agent behaviour anf can be *true* or *false*. Collective behaviour may be modelled implicitly via the individual agent behaviour, as is the case in RAPPID [19], or it may be modelled explicitly; for example, in Zeus it is modelled by role models [17]. Collective behaviours should be considered as first class design constructs enabling reasoning at a high abstraction level [12] and hence resulting to lower design complexity.

3. *Non-functional aspects:* This aspect refers to whether non-functional aspects are explicitly considered in the method and can be *true* or *false*. Non-functional aspects can be explicitly represented by appropriate modelling constructs, such as in Tropos [5], or they can be implicitly modelled within individual agent behaviour such as in Gaia [26]. 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 method 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 [26]. In the bottom-up perspective, design models are progressively composed from existing finer-grain models thus enabling reuse [12]. Supporting both perspectives, as in MESSAGE/UML [6], results in lower design complexity.

2. *Support for Reuse:* This aspect refers to whether the method 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 [17] there are guidelines for creating, storing and reusing negotiation strategies when specifying agent interactions, whilst in RAPPID [19] 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 method 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 [26]. However, other steps could be automated and carried out by a software tool, for example based on formal model transformations [21]. 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 [19] where there are no formal underpinnings. Automating the design process results in lower design complexity and reduces development effort and errors [1].

**Pragmatics**

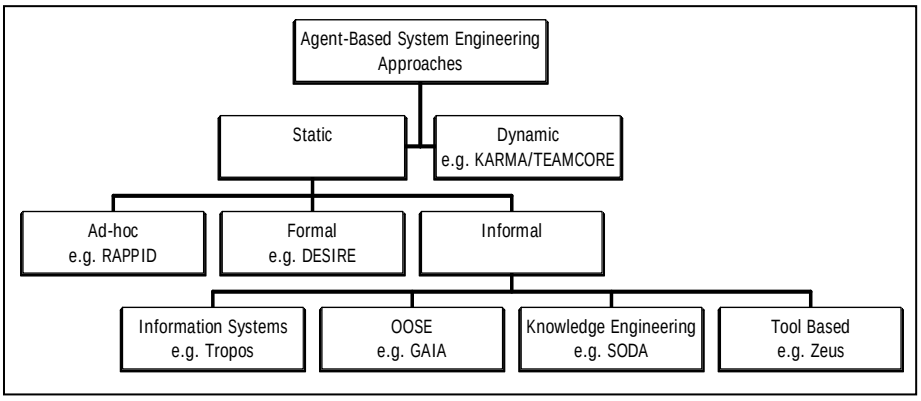
This view focuses on the pragmatics of each ABS engineering method. In other words, this view refers to how practical the method is for the design of real-world agent systems. It is concerned with the following aspects:

1. *Generality:* The generality of a method refers to the existence of restrictive premises concerning the environment and the application domain that affect the applicability of the method and can be characterised as *high*, *medium* or *low*. High generality means that the method can be applied without any significant restrictions, such as Tropos [5]. The generality is medium when there are considerable restrictions but the applicability of the method is still wide. For example, Gaia [26] assumes closed ABSs and small numbers of cooperating agents. In contrast, RAPPID [19] 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 method 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 methods, such as [12], 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 [5] 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 method. For example, the role-based approach method in [17] 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 [26] 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 [14] and automates repetitive tasks [16] increasing the usability of the method 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 [19]. However, this is not always the case, For example, Tropos [5] is bounded to only BDI agents and it is still applicable in many application domains.

**Fig. 2.** Classification of Agent-Based System Engineering Methods

## 3 Comparative Evaluation of ABS Engineering Approaches

ABS engineering methods can be classified as ad-hoc, formal, informal and structured, and dynamic (see Fig. 2). Ad-hoc methods involve designing an ABS in an application domain specific manner while formal approaches are based on the use of formal methods. Informal and structured methods originate from knowledge engineering and software engineering and are predominantly extensions of object-oriented analysis and design methodologies. Finally, dynamic methods 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 methods being regarded as more practical for numerous real-world applications.

A representative method (RAPPID [19], DESIRE [4], Gaia [26], MESSAGE/UML [6], Tropos [5], Zeus [17] and KARMA [22]) from each class of the above classification scheme has been evaluated according to the four views of 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 methods can be found in [11].

Regarding the *Concepts* perspective, about half of the ABS engineering methods examined (DESIRE, Tropos and Zeus) are bounded to a specific agent architecture. RAPPID is the only one limited to specific agent types as well. Furthermore, the majority of the methods examined (DESIRE, Gaia, Tropos, Zeus and KARMA) consider design as an explicit step in the ABS engineering lifecycle. However, only KARMA provides formal support for heuristics in the design of the ABS. Clearly, this is a general deficiency of current ABS engineering methods.

As far as it concerns the *Models* perspective, only Zeus and KARMA explicitly model organisational settings. Representing collective behaviours as first class design constructs is also not supported in most of the examined methods. 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.

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

**Table 1.** Evaluation of ABS engineering methods with respect to design complexity

In the *Process* perspective, only MESSAGE/UML allows working in both top-down and bottom-up fashion and the current version of MESSAGE/UML supports only the analysis phase of the ABS engineering lifecycle. Zeus supports bottom up design, the rest of the approaches are all allowing top-down design. Furthermore, only two approaches explicitly provide support for reuse, DESIRE and Zeus. DESIRE includes guidelines about how the agent system designer can reuse generic task components in the design of the ABS and Zeus includes guidelines about how to reuse generic behaviours represented by role models and generic agent characteristics — for example negotiation strategies. 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.

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. Zeus 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.

## 4 Implications for Further Research

The above analysis has demonstrated that none of the ABS engineering methods 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.

### 4.1 Support for Design Heuristics

Existing ABS engineering methods do not provide systematic and rigorous models for considering heuristics in the design of the ABS. In methods 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 methods support informal ABS design heuristics. For example in Zeus [17] 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.

### 4.2 Organisational Settings

Some ABS engineering methods explicitly model organisational settings — for example, MAS-CommonKADS [10] and SODA [18] — 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 [28]. Furthermore, organisational settings are not considered as first class design constructs apart from a few exceptions of approaches that use roles [17], [18]. Another problem concerning organisational settings is that existing approaches do not

provide rigorous methods for combining organisational settings with application functionality. This has to be done intuitively by the designer without any assistance by a software tool.

### 4.3 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], [12]. However, even where this is issue is addressed, such as in Zeus [17], there is no rigorous way to reuse collective application functionality and combine it with organisational settings.

### 4.4 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 other ABS engineering approach explicitly considers non-functional aspects in design apart from Tropos [5], which, at some stage, includes introducing actors and sub-actors that contribute positively to the satisfaction of non-functional requirements. 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 [22]. In these instances non-functional aspects are taken into account by adjusting the agent behaviour and the organisation of the ABS on 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 system instability.

### 4.5 Automating the Design Process

In order to reduce development effort and software design errors the design process should be partially automated [13]. This view is also adopted by informal ABS engineering methods [7], [21], [24] that try to provide the formal underpinnings for automatically designing ABSs from appropriate informal specifications. 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 [21] 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 [22]. However, KARMA assumes that agents already exist in cyberspace, which is not generally the case.

### 4.6 Working at Different Abstraction Levels

There is a consensus that abstraction in software design reduces design complexity [15]. 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 methods 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 [22]. 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 [17]. For example, in Zeus, 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.

## 5 Summary

This paper proposed a framework to assess ABS engineering methods with respect to design complexity they involve. Using this framework, a set of representative methods have 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 methods examined supports all aspects of the proposed framework. Therefore, new ABS engineering methods are needed providing better support for the framework aspects resulting thus to lower design complexity. 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. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: Modeling Early Requirements in Tropos: a Transformation Based Approach. 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. 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.
7. 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.
8. Fenton, N., Pfleeger, S.L.: Software Metrics: A Rigorous and Practical Approach. Boston, MA, USA: PWS Publishing Co., 1997.
9. 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.
10. 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.
11. 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.

12. Kendall, E.A.: Role models - patterns of agent system analysis and design. BT Technology Journal, 17, 4 (October 1999), 46-57.
13. Lowry, M.R., McCartney, R.D. (eds.): Automating Software Design. Menlo Park, CA: AAAI Press, 1991.
14. 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.
15. Metzger, A., Quelns, 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.
16. Ng, K., Kramer, J., Magee, J.: A CASE Tool for Software Architecture Design. Automated Software Engineering, 3, 3/4 (1996), 261-284.
17. 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.
18. 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.
19. 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.
20. 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.
21. 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.
22. Tambe, M., Pynadath, D.V., Chauvat, N.: Building Dynamic Agent Organisations in Cyberspace. IEEE Internet Computing, 4, 2 (March/April 2000), 65-73.
23. The Object Agency Inc.: A Comparison of Object-Oriented Development Methodologies. The Object Agency, Inc, (Autumn 1995), http://www.toa.com/pub/mcr.pdf.
24. 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.
25. Wooldridge, M.: On the Sources of Complexity in Agent Design. Applied Artificial Intelligence, 14, 7 (August 2000), 623-644.
26. 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.
27. 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.
28. 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.