Consistency Check in Modelling Multi-Agent Systems

Lijun Shan
Dept. of Comp., National Univ. of Defence Tech.
Changsha, 410073, China
Email: lijunshancn@yahoo.com

Hong Zhu
Dept. of Comp., Oxford Brookes Univ.
Oxford OX33 1HX, UK
Email: hzhu@brookes.ac.uk

Abstract

In model-driven software development, inconsistency of a model must be detected and eliminated to ensure the quality of the model. This paper investigates the consistency check in the modelling of multi-agent systems (MAS). Consistency constraints are formally defined for the CAMLE language, which was proposed in our previous work for modelling MAS. Uses of the consistency constraints in the implementation of a modelling environment for automatic consistency check and model transformation are discussed.

1. Introduction

Agent technology has been widely recognized as a viable solution for applications in dynamic environments such as the Internet [1]. A number of agent-oriented methodologies have been proposed in the literature, such as Gaia [2]. In [3, 4], we proposed a modelling language CAMLE, which stands for *C*aste-centric *A*gent-oriented *M*odelling *L*anguage and *E*nvironment, to support the development of multi-agent systems (MAS). Diagrammatic models in CAMLE serve as the base for the design and implementation of MAS, for example, to generate formal specifications in SLABS [5, 6].

CAMLE language is based on the principle of multiple views. Models from different views must be consistent with each other before they are utilized for further development. Inconsistency not only results in incorrect outcomes in the later stages, but also causes unnecessary complexity for developing tools that process the models. Therefore, inconsistency must be identified, managed and resolved. Models' consistency cannot rely on manual test, which is expensive, labour intensive and error prone. It is unacceptable especially for a modelling language like CAMLE that is designed to support evolutionary software development in which models are to be incrementally refined, extended, and revised through many cycles. In this paper, we investigate the automatic consistency check of MAS models by defining consistency constraints of the CAMLE language and implementing them as consistency check tools in the modelling environment.

The remainder of the paper is organized as follows. Section 2 briefly reviews the basic concepts used in CAMLE language. Section 3 defines the consistency constraints on CAMLE models. Section 4 discusses the uses of consistency constraints in the implementation of the modelling environment. Section 5 concludes this paper with a discussion of related works.

2. Overview of CAMLE

A number of theories and models of agent-based systems have been proposed in the literature; cf. [7]. CAMLE is based on the conceptual model of MAS formally defined in SLABS, which is a *s*pecification *l*anguage for *a*gent-*b*ased *s*ystems [5, 6].

In our model, agents are the basic entities of MAS. They are defined as real-time active computational entities that encapsulate data, operations and behaviours and situate in their designated environments. Therefore, by our definition, objects are degenerate forms of agents. Consequently, everything in a MAS is an agent. These agents are classified into a number of castes, which are agent classifiers. A caste is a set of agents that have the same structural and behavioural characteristics. Agents are instances of castes. However, an agent can change its casteship, viz. membership to a caste, by joining in a caste or retreating from its current caste at run-time. Inheritance (is-a relation), aggregation (whole-part relations), migration (role change relations) and collaboration are the basic relationships between castes. The environment of an agent in a MAS is a subset of the agents in the system. An agent communicates with others by taking visible actions and changing visible state variables as an information sender, and by observing other agents' visible actions and state variables as a receiver. The environment description of an agent or a caste defines which agents are visible; see [4, 5] for more details.

Compared to the concept of objects in object-orientation, the concept of agents in our definition highlights agents' features of encapsulation, autonomy and collaboration because behaviour rules and environment are specified in agent structure, changing casteship is in agents' ability and communication is through agents' visible activities rather than objects' method invoking. In comparison with other approaches in the research on agent-oriented software engineering such as FIPA [8] and AUML [9], we give a definition to conception of agent and MAS constructively rather than by a set of characteristics features.

In CAMLE, a MAS is specified with three types of models: caste models, collaboration models and behaviour models. Each model consists of one or more diagrams. The caste model describes the castes in the system and the structural relationships between them. A caste is a compound caste if it is composed of a number of other castes; otherwise, it is atomic. For each compound caste, a collaboration model and a behaviour model are constructed, while each atomic caste only has a behaviour model.

3. Consistency constraints

Here, consistency constraints refer to the conditions on the uses of diagrammatic notations, variables and names, types and symbols that a set of well-formed diagrams must satisfy so that they can be regarded as forming a meaningful model. These conditions are usually related to the semantics of the diagrams, but can be syntactically checked effectively and efficiently.

3.1. Caste models

We view an information system as an organization that consists of a collection of agents. The agents stand in certain relationships one to another by being a member of certain groups and playing certain roles, i.e. in certain castes. Such an organizational structure is captured in a caste model represented by a caste diagram; see Figure 1 for an example.

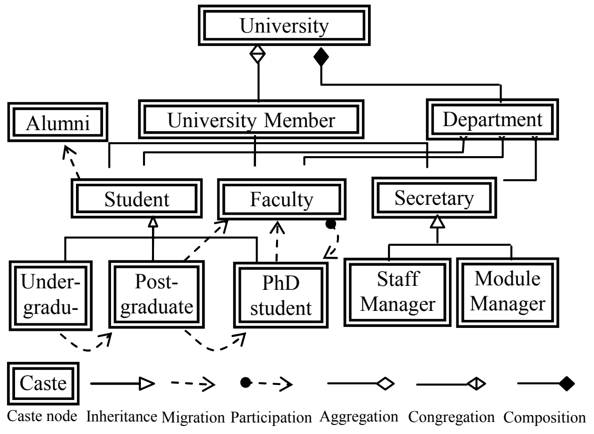

Figure 1. Example of caste diagram

A caste diagram defines the castes in the system, indicates three kinds of relationships between the castes: inheritance, aggregation and migration.

A well-formed caste diagram must satisfy the following conditions.

a) A caste diagram defines a naming space in which each node defines a caste with a unique name.
b) Each link defines a binary relation on castes by linking two nodes in the diagram.
c) An inheritance relation and a migration relation must be associated to two different caste nodes.
d) Inheritance relations must not form any loops.

Note that aggregation and migration relations are allowed to form loops. It is not required for an aggregation relation to be associated to different caste nodes.

3.2. Collaboration models

Collaboration models describe the dynamic structure of a system from communication perspective.

As shown in Figure 2, there are two types of nodes in a collaboration diagram (CD). In a CD, an agent node represents a specific agent while a caste node represents any agent in a caste. An arrow from node A to node B indicates that A's visible actions are observed by agent B. The actions with their parameters are annotated on the arrow.

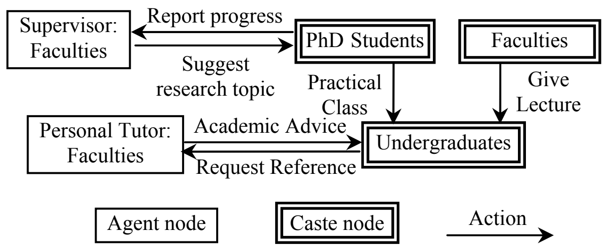

Figure 2. Example of collaboration diagram

A collaboration model may contain a number of CDs, including a general collaboration diagram (GCD) and a set of specific collaboration diagrams (SCD). A GCD serves as a declaration of what castes and their instance agents are involved in collaborations, while SCDs define the details of the collaboration protocols in various scenarios. Each SCD specifies a linear sequence of actions taken by the agents in a specific scenario of collaboration. CAMLE also supports the decomposition of an agent into a number of component agents in the same way as the analysis of the whole system. The collaboration among the component agents can also be defined by a collaboration model. Thus, a hierarchy of collaboration models for the system and all the agents can be formed. The following are the well-formedness conditions imposed on CDs.

a) Each caste / agent node must have a unique name.

b) The number assigned to an action indicating its temporal order must be unique, if any.

Let G be a GCD, $\boldsymbol{S}$ be the set of SCD and $D\in\boldsymbol{S}$ be any given SCD. Let $ANode(X)$, $CNode(X)$ and $Node(X)$ denote the set of agent nodes, the set of caste nodes and the set of all nodes in the CD X, respectively. Let $CName(x)$ denote the caste name of a node x. Nodes and linkages in G and those in $\boldsymbol{S}$ must satisfy the following consistency conditions.

c) Every agent node in the GCD G must appear in at least one SCD. Formally,

$$\forall n\in ANode(G).\exists D\in\boldsymbol{S}.\ (n\in ANode(D))$$

d) A caste node in the GCD must appear at least once in a SCD as either a caste node or an agent node representing a specific agent of the caste. Formally,

$$\forall n\in CNode(G).\exists D\in\boldsymbol{S}.\ (n\in CNode(D)\vee \exists n'\in ANode(D).(CName(n')=CName(n)))$$

e) Every caste node in a SCD must also appear in the GCD. Formally,

$$\forall D\in\boldsymbol{S}.\forall n\in CNode(D).(n\in CNode(G))$$

f) For every agent node in any SCD, there must be either a node of the same agent or the caste of the agent in the GCD. Formally,

$$\forall D\in\boldsymbol{S}.\forall n\in ANode(D).\ (n\in ANode(G)\vee \exists n'\in CNode(G).(CName(n')=CName(n))).$$

Assume that $a = ActName(p_1, p_2, \ldots, p_n)$ is an action associated to an arrow from node b to c. We call $<a, b, c>$ an interaction from b to c with action a. Let $Interaction(X)$ be the set of all interactions in a CD X. Let $\alpha=<a, b, c>$ be any given interaction. We write $Action(\alpha)$ for a, $Begin(\alpha)$ for b and $End(\alpha)$ for c.

g) Every interaction in a GCD must appear in at least one SCD, where a caste in the GCD can be replaced by an agent of the caste in the SCD. Formally,

$$\begin{gathered}\forall\alpha\in Interaction(G).\exists D\in\boldsymbol{S}.\exists\beta\in Interaction(D).\\ (CName(Begin(\alpha))=CName(Begin(\beta))\wedge\\ CName(End(\alpha))=CName(End(\beta))\wedge\\ Action(\alpha)=Action(\beta)\wedge\\ Begin(\alpha)\in ANode(G)\Rightarrow Begin(\beta)\in ANode(D)\wedge\\ End(\alpha)\in ANode(G)\Rightarrow End(\beta)\in ANode(D))\end{gathered}$$

h) Every interaction in an SCD must also be defined in the GCD. Formally,

$$\begin{gathered}\forall D\in\boldsymbol{S}.\forall\alpha\in Interaction(D).\ \exists\beta\in Interaction(G).\\ (CName(Begin(\alpha))=CName(Begin(\beta))\wedge\\ CName(End(\alpha))=CName(End(\beta))\wedge\\ Action(\alpha)=Action(\beta)\wedge\\ Begin(\alpha)\in CNode(G)\Rightarrow Begin(\beta)\in CNode(D)\wedge\\ End(\alpha)\in CNode(G)\Rightarrow End(\beta)\in CNode(D))\end{gathered}$$

Let X be a CD. We use $Env(X)$ to denote the environment of X, i.e. the set of agent and caste nodes on the boundary of X.

i) The environment of an SCD must be identical to the environment of the GCD. Formally,

$$\forall D\in\boldsymbol{S}.\ (Env(D)=Env(G))$$

For the sake of simplicity, we assume that a collaboration model (CM) satisfies the consistency constraints within one model discussed above. Therefore, we can overload the notation $Env(X)$ defined on diagrams to be the environment of the model, i.e. for a model M and any diagram D in M, define $Env(M) = Env(D)$, provided that M satisfies condition 3.2.i).

Let C be a compound caste in a CM M, and M_C be the CM for C, that is, M_C specifies the collaborations between C's components. The environment of C defined in M should be consistent with the environment description in M_C. The following two constraints are imposed on the models at different levels.

j) The set of agents and castes in C's environment described in M must be equal to the set of agents and castes in M_C's environment description. Formally,

$$\forall n.(n\in Env(M_C)\Leftrightarrow\exists\alpha\in Interaction(G).\ (n=Begin(\alpha)\wedge C=End(\alpha)));$$

where G is the GCD in M.

k) The interactions that C participates as an observer described in M must be realized as interactions between environment elements and C's components in M_C. Formally,

$$\begin{gathered}\forall\alpha\in Interaction(G).\exists\beta\in Interaction(G_C).(End(\alpha)=C\Rightarrow\\ Begin(\alpha)=Begin(\beta)\wedge Action(\alpha)=Action(\beta)\wedge\\ Begin(\beta)\in Env(G_C)\wedge End(\beta)\in Component(G_C));\end{gathered}$$

where G_C is the GCD in M_C and $Component(G_C)$ is the set of C's components depicted in G_C.

3.3. Behaviour models

Each caste is associated with a behaviour model (BM), which contains two kinds of diagrams: scenario diagrams (SD) and behaviour diagrams (BD).

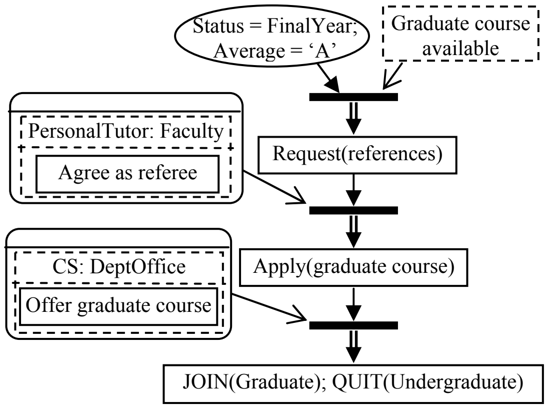

Figure 3. Example of behaviour diagram

An SD describes a typical situation in the operation of a system from an agent's view. SDs are referred to in BDs. A BD containing a number of behaviour rules describes an agent's designed behaviours in certain scenarios. Readers are referred to [4] for details of the notations and their semantics.

There are six different kinds of arrows that connect different kinds of nodes in BDs. In addition to the conditions for using right arrows between the nodes, the following well-formedness conditions are also imposed on BD and SD.

a) The temporal order between events must be linear, i.e. the in-degree and out-degree of an event node must be less than or equal to 1.
b) The logic connective nodes 'AND' and 'OR' are binary operators, and 'NOT' is unitary operator.
c) A transition bar has at most three nodes directly connected to it: at most one scenario, at most one pre-condition node, and at most one event node.

Each scenario reference node in a BD refers to a scenario defined in a SD. Therefore, a consistency condition on the relationship between a BD and the SDs in one behaviour model is defined as follows.

d) The set of scenarios referred to in a BD by using scenario reference nodes is a subset of the scenarios defined by SDs. Formally, let C be a caste, D_C be the BD of caste C, and $\boldsymbol{S}_C$ be the set of SDs of C.

$\forall n\in ScenarioNode(D_C).\ \exists S\in \boldsymbol{S}_C.(Name(n)=Name(S)$

3.4. Consistency between models

All the constraints defined above are intra-model constraints since they are on a same type of models. This subsection discusses the consistency between different types of models and defines inter-model constraints. In the sequel, models are assumed to be consistent with regard to the intra-model constraints.

3.4.1. Between collaboration model and caste model

Let ***CD*** be the set of CDs in a collaboration model (CM) and C the caste model for the system in question.

a) The set of the castes in the CM must be a subset of the castes in caste model. Formally,

$$\forall D\in \boldsymbol{CD}.\forall n\in Node(D).\exists n'\in Node(C).$$
$$(CName(n) = Name(n'))$$

It is possible that a caste in the caste model does not appear in any CD. For example, a caste can be an abstract caste, which has no direct instance agent and any instance of the caste is always an instance of its sub-caste. The behaviours of the agents of the abstract caste can be defined by its sub-castes. Consequently, the abstract caste may not occur in any CD.

Let ***CM*** be the collection of collaboration models of the system. Let x be a caste in the system, and M_x be the collaboration model for x. For models M_A and M_B in ***CM***, we say that M_B is an *immediate refinement* of model M_A and write $M_B \lhd M_A$, if B is the component caste of caste A. Let $Aggr(C)$ be the set of aggregation relations in the caste model C.

b) The hierarchical structure of the CMs must be consistent with the whole-part relations between castes defined in caste diagram. Formally,

$$\forall M_A,M_B\in \boldsymbol{CM}.(M_B \lhd M_A\Rightarrow\exists R\in Aggr(C).(R(B,A))$$

3.4.2. Between behaviour model and caste model

Let ***BM*** be the set of behaviour models (BM) of a system, and C the caste model. The caste with a BM X defining its behaviour is denoted by $Caste(X)$.

c) Each BM defines the behaviour of a caste and the caste must be in the caste model. Formally,

$$\forall B\in \boldsymbol{BM}.\exists n\in Node(C).(Caste(B)= n\ ).$$

In a BM, say, of caste B, the description of scenarios may refer to the agents in the environment of B. Let $Agents(B)$ be the set of agents referred to in a BM B, $Caste\text{-}of(x)$ the caste of such an agent.

d) Every agent in a scenario in a BM must have its caste defined in the caste model. Formally,

$$\forall B\in \boldsymbol{BM}.\forall a\in Agents(B).\exists n\in Node(C).$$
$$(Caste\text{-}of(a)=Name(n)).$$

In a caste model, an agent's change of casteship is described through a migration relation between the castes. In a BM, an agent's change of casteship is defined through actions *JOIN*(*caste*), *MOVETO*(*caste*) and *QUIT*. Such information in the BM must be consistent with the caste model.

e) Let B_C be the BM for caste C.
   - B_C contains an action *JOIN*(*C*'), where *C*' is a caste name, if and only if there is a participation relation from *C* to *C*' in the caste model.
   - If B_C contains an action *MOVETO*(*C'*), where *C'* is a caste name, there must be a migration relation from *C* to *C'* in the caste model.
   - If B_C contains an action *QUIT*, there must be a migration relation from *C* to some caste in the caste model.
   - If there is a migration from *C* to some caste (say *C'*) in the caste model, there must be either a *MOVETO*(*C'*) or *QUIT* action in the BM of *C*.

By '*an action in a behaviour model'*, we mean a result action of a behaviour rule, depicted as an action node immediately after a transition bar in a BD.

3.4.3. Between collaboration model and behaviour model

Let *Components(C)* be the set of *C*'s component castes. Let *VisibleActions*(*C*) be the set of visible actions of caste *C* defined in the CM. Let B_X be the BM for caste *X*, *Rules*(*B*) be the set of rules in BM *B*, and *Action*(*r*) be the result action of rule *r*.

f) Every visible action of caste *C* defined in the CM must occur in the BM of *C* or at least one of *C*'s components as a result action. Formally,

$\forall a \in VisibleActions(C).(\exists r \in Rules(B_C) \vee$
$(\exists M \in Components(C).\ \exists r \in Rules(B_M)).\ (a = Action(r))$

Let G be a caste or agent that has a communication link to caste C in the CM. We call G a collaborator of caste C, and write $Collaborators(C)$ for the set of C's collaborators. Let $Scenarios(B)$ be the set of scenarios used in a BM B, and $Ref(Sc)$ denote the set of castes or agents that a scenario Sc refers to.

g) For each scenario used in the definition of caste C's behaviour, the agents and/or castes that the scenario referred to must occur in the CM as C's collaborators. Formally,

$\forall \mathrm{Sc} \in Scenarios(B_C).\forall G \in Ref(Sc).G \in Collaborators(C)$

A scenario actor may be specified with qualifier, e.g. '$\forall A$:*CasteX*', and '$\exists Y$:*CasteX*'. The caste *CasteX* must be a collaborator of caste C. If the actor of a scenario refers to a specific agent, i.e. in the form of '*AgentM*:*CasteX*', the agent *AgentM* of caste *CasteX* must be a collaborator.

h) The agents and castes referred to in a scenario must be elements in the environment of the caste described by the CM. Formally, let C to be the caste described by a BM B.

$$\forall Sc \in Scenarios(B).\forall G \in Ref(Sc).(\mathrm{G} \in Env(C)).$$

The collaboration between an agent A of caste C and other agents may be realized through the collaboration of A's component agents. Therefore, we do not require all collaborators of caste C to be referred to in the definition of caste C's behaviour.

Let p_1, p_2, …, p_n be the sequence of actions of a caste C (or an agent of caste C) described in a scenario Sc. Each of p_i, i=1, 2, …, n, is called a referred action of caste C in scenario Sc. We write $ReferredActions(C, Sc)$ to denote the set of all such actions.

i) Every referred action in a scenario used in a BD must be a visible action of the caste described by the scenario. Formally,

$$\forall \mathrm{Sc} \in Scenarios(B_C).\forall a \in ReferredActions(C, Sc).$$
$$(a \in VisibleActions(C)).$$

It is not required that all visible actions of a collaborator should be referred to in the definition of a caste's behaviour, because the collaboration may be realized through component agents.

4. Uses of consistency constraints

Consistency conditions can play at least two important roles in model-based development. First, they serve as check points for quality assurance in modelling process. Violation of the conditions indicates the existence of contradictions in the model. Inconsistency may also be caused by conflict in requirements. Consistency checks on requirement models help to identify and thereafter to resolve and manage such conflict. Therefore, automatic consistency check can help engineers to detect errors at modelling stage, hence prevent errors from being propagated to later stages. Second, in model-driven development of software systems, it is desirable to automatically transform one model to another model, and to generate code (or code framework) from models. Design and implementation of such tools must ensure that the transformation rules preserve the models' meanings. Consistency conditions provide a means to formally specify the correctness of the transformation rules.

The consistency constraints defined in this paper have been used for both of the above purposes in the implementation of CAMLE environment [4]. They are computable and have been directly implemented in the environment as consistency check tools. Diagnostic information as the result of the check is reported to users to help locate and correct errors. A diagram generator in the environment generates partial models (incomplete diagrams) from existing diagrams to help model construction. The rules to generate partial models are based on the consistency constraints so that the generated partial diagrams are consistent with existing ones. Preliminary case studies show that both consistency check and partial model generations are very helpful to improve the models' quality and software engineers' productivity. The case studies are omitted here for the sake of space. Besides model construction and checking, another main function of CAMLE environment is to automatically transform a model into the system's formal specifications in SLABS. Consistency check simplifies the error processing in the implementation of the automatic transformation tool.

5. Conclusion

In this paper, we defined the consistency constraints on CAMLE models and investigated its uses in the modelling environment. The following table summarizes and classifies the consistency constraints.

Table 1. Classification of Constraints

		Horizontal Consistency	Vertical Consistency	
			Local	Global
Intra-model	Intra-diagram	1a, 1b, 1c, 1d, 1e, 2a, 2b, 3a, 3b, 3c	–	–
	Inter-diagram	2c, 2d, 2e, 2f, 2g, 2h, 2i, 3d	2j, 2k	–
Inter-model		4f, 4g, 4h, 4i	4e	4a, 4b, 4c, 4d

The constraints in Table 1 are referred to by their numbers in section 3. For example, 2b refers to condition b) in section 3.2. Note that the horizontal / vertical consistency means the consistency of models on same / different abstract level. Local consistency conditions are imposed on models of adjacent levels while global consistency conditions are concerned with the whole hierarchical structure of the models.

Well-defined visual notations for modelling software systems' structures and behaviours have the advantages of readability and preciseness due to their semi-formal nature. A common feature of such visual notations is that multiple views are utilized to model a system's different aspects and/or at different levels of abstraction. The consistency between various views is crucial for developing quality software systems. It is, therefore, desirable to automatically check the consistency among the diagrams [10, 11]. The past few years has seen a rapid increase in the research on defining consistency conditions and implementing consistency check tools for modelling languages, especially for UML [12, 13, 14, 15]. However, defining consistency between different views or diagrams is not trivial [16]. Most existing modelling languages, for example UML, have no explicitly defined consistency constraints.

Among the related works on consistency check, Xlinkit is a flexible tool for checking the consistency of distributed heterogeneous documents [17]. It comprises a language for expressing constraints between such documents, a document management mechanism and an engine that checks the documents against the constraints. In comparison with Xlinkit, our approach is language specific. The direct implementation of consistency constraints as a part of modelling environment is highly efficient and effective in detecting errors. In addition, the explicitly defined constraints form a base for automatic transformations between models. Formal methods, such as model checking, have also been used for checking the consistency between multiple views of software specifications, e.g. in [18, 19]. It requires translating models into a formal notation as the input to a model checker, while assumes that syntactic errors have been removed before the translation. Therefore, to check consistency before translation is still necessary. As formal specifications are automatically generated from a consistent model by our tools, application of formal methods to analyse the models becomes possible. This is a direction for our further investigation.

Acknowledgement

The work reported in this paper is supported by China High-Technology Research and Development Programme under the grant 2002AA116070.

References

[1] Jennings, N.& Wooldridge,M. (Eds.), *Agent Technology: Foundations*, Applications And Markets, Springer, 1998.

[2] Zambonelli, F., Jennings, NR & Wooldridge, M. Developing multiagent systems: the Gaia Methodology. *ACM Trans on Software Eng. and Meth.* 12(3): 317-370, 2003.

[3] Shan, L. & Zhu, H. Analysing and specifying scenarios and agent behaviours. *Proc. of IEEE/WIC Int. Conf. on Intelligent Agent Technology.* Halifax, Canada. 2003.

[4] Shan, L. & Zhu, H. CAMLE: A Caste-Centric Agent Modelling Language and Environment. *Proc. of the 3rd International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS 2004*) at ICSE 2004, Edinburgh, Scotland (UK), May 2004.

[5] Zhu, H. SLABS: A Formal Specification Language for Agent-Based Systems, *Journal of Software Engineering and Knowledge Engineering* 11(5), 529-558, 2001.

[6] Zhu, H., A formal specification language for agent-oriented software engineering, *Proc. of AAMAS'2003*, Melbourne, Australia, July 2003.

[7] Wooldridge, M. J. & Jennings, N. R. Agent theories, architectures, and languages: a survey. *Intelligent Agents: Theories, Architectures, and Languages*, LNAI 890, 1-32, Springer-Verlag, 1995.

[8] FIPA. http://www.fipa.org

[9] AUML. http://www.auml.org

[10] Xu, J., Jin, L., & Zhu, H. Tool support of orderly transition from informal to formal descriptions in requirements engineering. *Proc. of IFIP'96: Advanced IT Tools*, Terashima, N. & Altman, E. (Eds.), Chapman & Hall, 199-206, 1996.

[11] Kuzniarz, L., Reggio, G., Sourrouille, J. L., & Huzar, Z. (eds.) Consistency Problems in UML-based Software Development, Workshop Materials at UML'2002, Research Report. Blekinge Institute of Technology, 2002.

[12] Pap, Z. S., Majzikl, I., Pataricza, A, & Szegi, A. Completeness and Consistency Analysis of UML Statechart Specifications. *Proc. of IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop,* 83-90. 2001.

[13]Andr´e, P., Romanczuk, A., Royer, J-C. Check the Consistency of UML Class Diagrams Using Larch Prover. *Proc. of 3rd Rigorous Object-Oriented Methods Workshop*, Clark T., (ed.), BCS, 2000.

[14] Paige, R. F., Ostroff, J. S., and Brooke, P. J. Check the Consistency of Collaboration and Class Diagrams using PVS. *Proc. of 4th Workshop on Rigorous Object-Oriented Methods*, London, British Computer Society, 2002.

[15]Astesiano, E. & Reggio, G.. An Attempt at Analysing the Consistency Problems in the UML from a Classical Algebraic Viewpoint. *Recent Trends in Algebraic Development Techniques, Selected Papers of the 15th Int. Workshop WADT'02*, LNCS, Springer Verlag, 2003.

[16]Nentwich, C., Emmerich, W. & Finkelstein, A. Static Consistency Check for Distributed Specifications. *Proc. of 16th Int. Conf. on Automated Software Engineering,* Coronado Island, CA., 115-124. 2001.

[17]Nentwich, C., Emmerich, W., & Finkelstein, A. Flexible Consistency Check. *ACM Transactions on Software Engineering and Methodology* 12 (1), 28-63, 2003.

[18]Inverardi, P., Muccini, H., Pelliccione, P. Automated check of architectural models consistency using SPIN. *Proc. of 16th IEEE Int. Conf. on Automated Software Engineering,* San Diego, California, p.346, 2001.

[19] Schafer, T., Knapp, A., & Merz, S. Model Check UML State Machines and Collaborations. *Workshop on Software Model Check,* Paris, July 2001.