Modeling and Simulating Adaptive Multi-Agent Systems with CAMLE

Lijun Shan, Chenglie Du
College of Computer Science,
Northwestern Polytechnical University,
Xi'an, China
lijunshancn@yahoo.com

Hong Zhu
Dept of Computing and Communication Technologies,
Oxford Brookes University,
Oxford, UK
hzhu@brookes.ac.uk

Abstract—**With the advent of embedded and mobile computing techniques, software systems are increasingly operated in open and dynamic environments. Such systems desire self-adaptive capabilities. This paper proposes an agent-oriented approach to the modeling and simulation of distributed adaptive systems. The approach enables the designers to construct structural and behavioral models at a high abstraction level, and to validate the models at design stage through simulation. The modeling language supports devising decentralized adaption logic that enables various types of adaptions at both component and system levels. The simulation of adaptation processes can demonstrate how a system dynamically re-organizes in response to the changes in the operating context. Case studies show that our approach can support various self-adaptation mechanisms in which a multi-agent system adapts to internal failures, unexpected environment, or user's changing requirements.**

Keywords -- self-adaptive systems, multi-agent systems, modeling, simulation

I. INTRODUCTION

With the advent of embedded and mobile computing techniques, software systems are increasingly operated in open and dynamic environments. Self-adaptive capabilities enhance such systems' reliability and robustness in their ever changing environments. A system is adaptive if it is able to adjust its structure and behavior at run time so as to recover from internal failures, to work in unexpected environment, or to meet user's changing requirements.

Agent-oriented paradigm, which regards a software system as a set of interacting autonomous computing elements (i.e. agents), is a promising approach to the construction of decentralized self-adaptive systems [1]. A multi-agent system with self-adaptive features is called an Adaptive Multi-Agent System (AMAS) [2]. The recent years have witnessed a rapid growth in the AI-related research on AMAS, which endeavors to improve agents' adaptability with more expressive logic and higher reasoning capabilities [3, 4]. However, how to engineer AMAS remains an open problem. In the framework of an agent-oriented software engineering (AOSE) paradigm, this paper advances a model-driven approach based on our previous work of modeling MAS.

Simulation of models is particularly valuable for AMAS development, verification and validation, because self-adaptation emerges at system level dynamically. Neither modeling nor simulation of AMAS is an easy task. The model has to describe how a system adapts at both the component level and the organization level, while the adaption logic scatters in various agents instead of centrally controlled. Simulation should enable the designer to tell whether a large number of agents can collaboratively achieve the adaption requirements.

The existing work on modeling AMAS can be classified into two categories. In the first category, existing agent-oriented software development methodologies for MAS are extended to meet the requirements of developing AMAS, such as Gaia [5], O-MaSE [6], ADELET [2] and SOTA [7]. These methodologies, however, provide neither language facilities for expressing designs of adaption at component level and system level, nor techniques for evaluating the designs. Researches in the second group propose conceptual models of AMAS based on the notion of organizations, e.g. LAO (Logic for Agent Organization) [4], reorganization protocol stack [3]. The existing formalisms of reorganization are mostly action-based, which works better on planning and reasoning than on software development.

This paper presents an approach for modeling and simulating AMAS with CAMLE, a caste-centered agent-oriented modeling language and environment proposed in our previous work [8, 9]. We report a series of case studies conducted to demonstrate how to model various self-adaption mechanisms in the CAMLE modeling language, and how to use the simulator tool of the CAMLE modeling environment in the verification of self-adaptive behaviors against various requirements.

II. THE CAMLE LANGUAGE AND ENVIRONMENT

CAMLE is a multiple-view graphic modeling language based on a consistent conceptual model of multi-agent systems.

A. The Conceptual Model

With CAMLE, an AMAS is modeled as a collection of agents that coordinate with each other, monitor the changes in the environment, and adapt their behaviors and organizational structure in a distributed setting. The main concepts of the CAMLE language are summarized in Figure 1.

For example, a rescuer robot can be defined as a caste named Robot, as shown in Figure 2. Agents can be created as instances of the caste when the parameters are given

specific values: integer values are assigned to *id* as the identifier, to *iStartX* and *iStartY* as the initial coordinates. A set of attributes represents the knowledge/state of the agent. For example, *iPosX* and *iPosY* represent a robot's current position. A set of actions enables the agent to perform certain tasks. For example, *Walk*() enables a robot to change its own location.

```
Agent = <Caste Instance>
Caste = <Parameters, Attributes, Actions, Environment,
          Behavior_Rule*>
Environment = <Agent*>
Interaction(A, B) = <A.Observation, B.PublicEvents>
Behavior  Rule = <Scenario, Agent State, Action*>
```

Figure 1 The conceptual model of CAMLE

```
CASTE Robot (int id, int iStartX, int iStartY)
{
    VAR: public:    int iPosX; int iPosY; int iDirection;
           int iFindVictimID; int iCurrentTask;
                    int allFoundVictimList[];
    ACT:  internal: void SetTask(int i);
}
```

Figure 2 Caste Robot

The concepts of role and organization have long been used in agent-oriented methodologies. In CAMLE, a caste can represent a role, an organization, etc. For example, the roles that an agent *A* plays can be defined as the sub-castes of *A*'s caste. A sub-caste inherits all the attributes and actions from its super-caste, but carries new behavior rules, additional state variables and actions that are specific to the role. In the example of rescuer robotics, a Hierarchy organization contains Leader and Member as two sub-castes of Robot.

An adaptive agent needs to change its role(s) at run time. For example, in the Hierarchy organization, when a Leader breaks down, a Member may change its role to Leader in order to keep the team working. Such changes of roles are naturally supported by CAMLE's caste facility that enables an agent to dynamically change its casteship (i.e. membership to castes) by taking a *join*, *quit*, *suspend* or *resume* action [10].

A system can accomplish a task through different organizations, and may change its organization at run-time to adapt to the context, though non-functional properties (such as efficiency and cost) may vary. For example, for the Searcher Robotics, a transfer from Hierarchy to Peer may enlarge the coverable area at the cost of higher communication traffic load.

B. The Graphic Notation

CAMLE provides three models, namely structure model, collaboration model and behavior model, to facilitate users to capture a system from different perspectives.

1) Structure Model

A structure model, consisting of a number of caste diagrams, gives an overview of the modeled AMAS by depicting castes, organizations and casteship transitions.

As shown in Figure 3, the Searcher Robotics team can take either Hierarchy or Peer organization. The caste Hierarchy consists of three castes: User, Leader and Member, where the latter two are sub-castes of Robot. Peer consists two castes: User and Rescuer, where the latter is also a sub-caste of Robot. The system can transfer its organization from Hierarchy to Peer, or vice versa if necessary. A Member agent may become Leader under certain circumstances. Each caste's structural elements, including parameters, attributes, actions and environment are also specified in the caste model. The function body of the actions can be written in C code for the CAMLE simulator to execute.

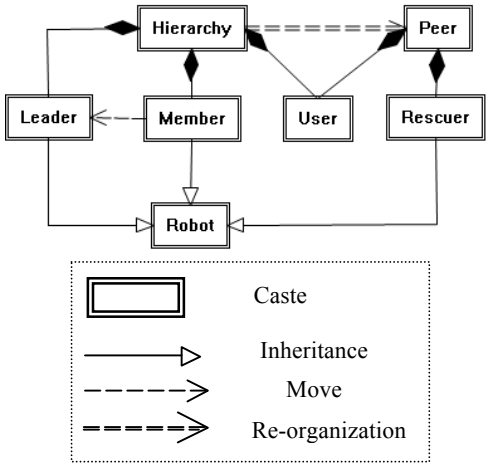


Figure 3 Caste diagram: example and notation

2) Collaboration model

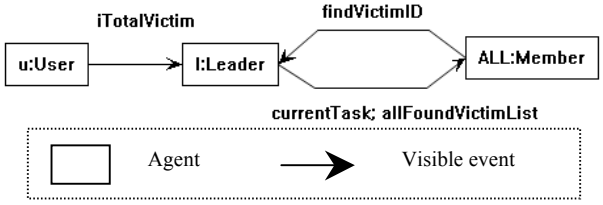


Figure 4 Collaboration diagram: example and notation

A collaboration model, consisting of a number of collaboration diagrams, describes typical interaction scenarios. As shown in Figure 4, in the Hierarchy organization, once the user assigns a value to its attribute *iTotalVictim*, which represents the total number of victims to find, the Leader robot observes that and assigns its own attribute *currentTask* accordingly. Then, every Member acquires the task by observing the Leader's attribute *currentTask*. The Leader constantly observes each Member's attribute *findVictimID*, which represents the recently found victim, to form a global view on the task's

progress. Each Member updates its record of the task status by observing the Leader's attribute *allFoundVictimList*.

3) Behavior model

A behavior model defines the behavior rules of each caste in an AMAS. A behavior rule consists of three parts: a scenario description about the state of the environment, the owner agent's current state, and the action(s) to take once the specified scenario and state are all satisfied.

For example, Figure 5 gives the graphic notation of CAMLE's behavior model and shows two behavior rules of the caste Leader. Figure 5 (A) states that if the value of the User's attribute *iTotalVictim* is greater than the value of the agent's own attribute *currentTask*, the Leader agent updates its task accordingly. Figure 5 (B) states that when the Leader observes that a Member has found a new victim (the value of the Member robot's attribute *findVictimID* is neither -1 nor within the Leader's record of the found victims), the Leader adds this victim's ID to its record of the found victims.

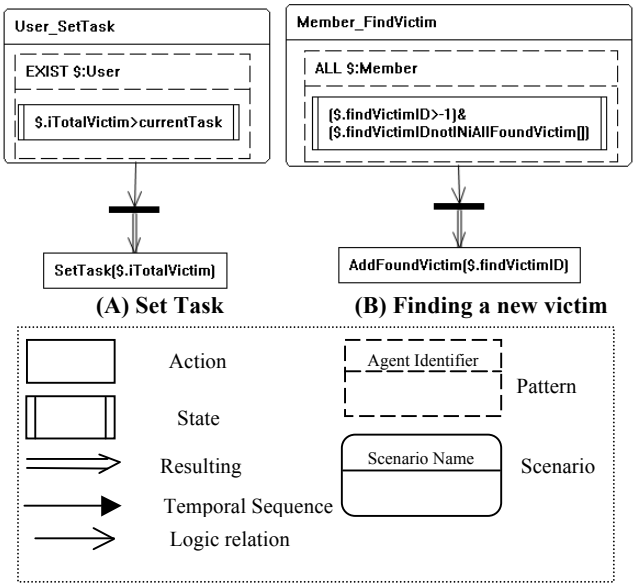


Figure 5 Behavior diagram: example and notation

C. Simulation Tool in CAMLE

The CAMLE modeling environment presented in [8] contains a set of tools for model construction, model consistency checking and model to formal specification transformation. To provide better support to the development of AMAS, we have further developed a dynamic simulation tool that interprets the graphic model. In addition to the diagrams, the simulation tool also takes a specification of system's initial configuration as input, which is written in C programming language. The specification defines global variables and organizations that are employed in the system, and describes the system's initialization. Each organization is specified by a function which defines how agents join castes by performing the primitive action *MOVE*(). The term *A.MOVE*(*C*) means that an agent *A* becomes a member of caste *C* and quits its previous caste if there is.

III. Case Study

To demonstrate how the CAMLE method supports modeling and simulating of AMAS, we conduct a series of experiments following the types of reorganization activities as classified in [4]. The system under study is a Searcher Robot team which can adapt to various situations.

A. Ideal Situation

A team of *N* robots is deployed to find a number of victims distributed over an arena shown in Figure 6, where robots are represented by green round dots, and victims are red square spots.

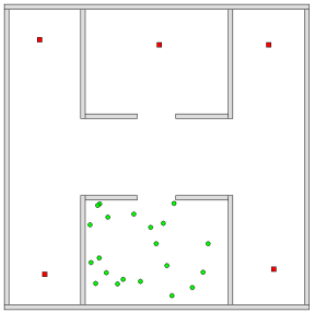


Figure 6 The arena for searching

The arena can be a dangerous place such as firing building or battle field. Ideally, the arena to explore is small enough for a robot to communicate with the other robots; the wireless communication condition is good enough to withstand any traffic load; the arena is so favorable that all robot works well all the time; the task is fixed once assigned to the robot team. In this situation, both Hierarchy and Peer organizations perform well. The results of the simulation with different configurations are summarized in Table I, where *P*(*N*) represents a Peer organization with *N* robots, and *H*(*X*, *Y*) represents a Hierarchy organization with *X* Leaders and *Y* Members. The timeout limit of an execution is set to be 500 seconds in the sequel.

TABLE I PERFORMANCE IN THE IDEAL SITUATION

		P(2)	P(21)	H(1, 1)	H(1, 20)
1st	**Min**	14605	3424	12897	2320
	Max	99795	13504	141686	10509
	Avg	52286	7062	65330	6648
5th	**Min**	97561	19749	223751	17241
	Max	Timeout (1)	58724	Timeout (5)	99988
	Avg	310192	33408	373192	39785

Table I gives the time (millisecond) spent to find the first and the last victim in 10 executions of each model, where the row *Avg* shows the average time which counts only the non-timeout values. Table I shows that the searching process is quicker with more robots. Given the same number of total robots, the Hierarchy organization is slower than Peer, because Hierarchy needs one Leader robot, which does not participate in searching victims.

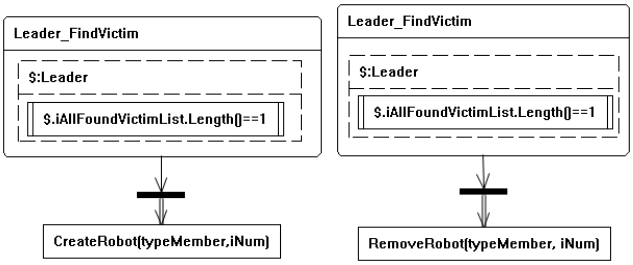
B. *Staffing*

Staffing is a simple type of re-organization in which the number of agents changes [4]. Two activities can be taken: *staff+*: adding new agents to the system, and *staff-*: deleting agents from the system. Suppose that the initial organization is *H*(1, 20), i.e. with 1 Leader and 20 Members. The following cases are considered, respectively.

(1) *Stable*: The organization keeps unchanged.

(2) *Staff+*: An User agent, with the behavior rule shown in Figure 7 (A), brings 10 more robots as Members when the first victim is found. This is modeled by one behavior rule. The action *CreateRobot*() in the behavior rule firstly instantiates new agents from the caste Robot, then attach them to the caste Member by taking the action *MOVE*(*Member*).

(3) *Staff-*: An User agent, with a behavior rule shown in Figure 7 (B), takes 10 member robots out of the team when the first victim is found. The action *RemoveRobot*() in the behavior rule deletes an agent *A* from the system by taking the action *DEL*(*A*).



(A) Adding Member robots (B) Removing Member robots

Figure 7 User's behavior rules for

The results of the models' simulation, as summarized in Table II, show that the task's average finish time drops when robots are added during the execution, and the time rises if robots are removed at run time.

TABLE II SEARCHER ROBOTICS: STAFFING

		H(1, 20)	H(1, 20) =>H(1, 30)	H(1, 20) =>H(1, 10)
1st	Min	2164	4010	3632
	Max	11030	13741	8795
	Avg	5551	6804	5232
5th	Min	19369	14313	14275
	Max	91924	42234	98535
	Avg	36604	23603	48841

C. *Re-staffing*

Re-staffing means relocating agents to other existing roles within the organization [4]. Three activities may happen: *enact*: assigning a new role to an agent; *deact*: removing a role from an agent; *move*: combining the above two.

In CAMLE, an agent *deacts* a role by taking an action *QUIT*(). In particular, if an agent has only one caste, deacting results in an inactive agent. An agent joins a caste by taking an action *JOIN*(). However, consistency between different the new role and the old role(s) must be ensured during the design. For instance, if a designer specifies that a leader robot joins the caste Member without quitting Leader, the behavior rule "moving around" of the caste Member contradicts with the behavior rule "staying at the initial position" of the caste Leader. Such contradictions may be not easily detected at the design stage, and will cause problems during the system's execution. The simulator helps to solve this problem.

Suppose the Searcher Robotics team initially has the structure *H*(1, 20). When the environment is adverse, the robots has a high probability of breaking down. For example, in a fire field the robots are exposed to a high temperature, or in a battlefield the robots may be captured and destroyed by the enemies. Assume that the robots break down with a possibility of 0.1% per 0.1 second. We construct models for the following two cases:

(1) *No role change*: No robot can change its role. Once the Leader breaks down, the team stops working.

(2) *Member becoming Leader*: On detecting the Leader's failure, a live Member robot becomes a Leader, and stays at its current position. This is realized through a behavior rule of the Member caste, as shown in Figure 8. This case illustrates that an agent can change its role when some failure occurs in the system.

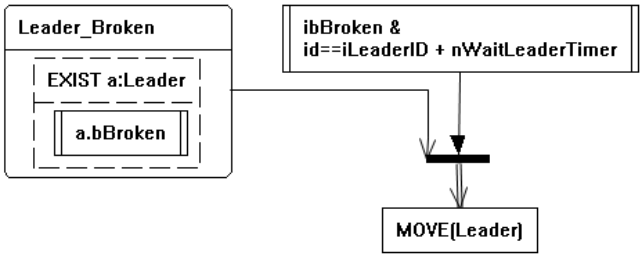


Figure 8 Member's behavior rule for becoming Leader

As summarized in Table III, without automatic role changing, no victim is found in two executions, because the Leader breaks down before the team finds any victim. In comparison, when the members are able to taking new roles, the task can be finished in all the ten executions.

TABLE III H(1,20) ROBOTS MAY BREAK DOWN

		No role change	Member->Leader
1st	Min	5171	3590
	Max	Timeout (2)	13261
	Avg	6937	7826
5th	Min	14672	14941
	Max	Timeout (4)	89849
	Avg	56863	44065

D. *Structuring*

Structuring means the system's structure is reshaped [4]. It involves four types of activities: *position+*/*position-*: to add/remove a position (i.e. role); *struct+*/*struct-*: to add/remove dependencies between existing positions.

Since *struct+*/*struct-* changes the interaction relationship between a specific pair of roles, while the other agents are not affected, the two activities can be modeled as agents' caste shifting.

So far, the above cases can be regarded as component-level adaption: staffing changes the number of agents of a certain caste, re-staffing changes an agent's casteship, and *struct+*/*struct-* changes the dependencies between some castes. Such re-organization activities neither alter the interaction protocol of the organization, nor change the behavior of other agents. In contrast, *position+*/*position-* affects the rest of the organization. When a new caste is added, or an existing caste is removed, the other agents' behaviors are possibly affected. Therefore, they have to alter their casteships accordingly.

In the Searcher Robotics example, suppose the arena is a square whose side length is 2**S* (*S* denotes a robot's communicational distance). Initially, the user, knowing neither the arena's area nor the wireless communication condition, assigns the hierarchy organization *H*(1, 20) to the team. We consider the following three situations:

(1) *No re-organization*: The team cannot change its organization. Once a Member cannot see the Leader, it returns along its passed route until it can see the Leader again.

(2) *Hierarchy to Peer:* Once the Leader finds that a Member robot goes too far to communicate with, it alters the system's organization to Peer. This is realized through the Leader taking an action *Peer*(), as shown in the behavior rule in Figure 9 (A). This case illustrates that an agent can initiate the system's organization change to adapt to the unexpected environment.

(3) *Hierarchy to Hybrid*: Once a Member goes far enough (with one more step it cannot see any Leader), it becomes a Leader. Consequently, the organization becomes *H*(*X*, *N-X*), and *X* may increase during the execution. This is realized through a behavior rule in the Member caste, as shown in Figure 9 (B). This case illustrates that an agent can autonomously change its casteship due to some change in its own state (a variable indicates whether it can communicate with a Leader), which effectively leads to re-organization of the system.

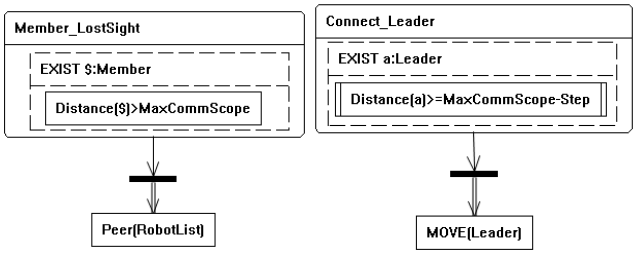


Figure 9 behavior rule for re-organization

The simulation results of the above models are summarized in Table IV. Among the three different organizations, the Peer structure can cover the biggest area. When some Member robots become Leader, the searching area is enlarged, but still some victims are not found within the time limit. Since Leaders do not move, increasing Leader robots reduces the number of Member robots. Consequently, the team needs to spend more time to find all the victims.

This case also shows that adaption can be achieved through different re-organization mechanisms.

TABLE IV THREE MODELS FOR BIG ARENA

		No change	H(1, 20) -> P(21)	H(1, 20) -> H(X, 21-X)
1st	Min	3528	2460	3644
	Max	28989	20034	32218
	Avg	15018	6966	12848
5th	Min	Timeout	9690	24123
	Max	Timeout	45074	Timeout (6)
	Avg	Timeout	26232	34001

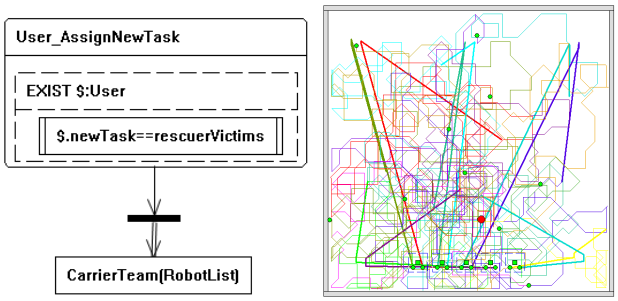
E. Strategy and Duty

A change in an organization's strategy and duty means the objectives of the organization is altered [4]. Here, we classify such changes into two types: (a) modifying some parameters of the objectives or tasks, (b) adding/removing certain objectives or tasks.

For the Searcher Robotics system, the first type of changes in systems' objectives may happen when the user changes the task's parameter at run time. For example, initially the user assigns a task of finding 3 victims to the robot team. Later on, when new information emerges, the number of known victims increases to 5. Then, the user re-assign task to the team to finding 5 victims. The behavior rule of the Leader robot given in Figure 5 (A) states that the robot constantly reads the user's command at run time. Similarly, all Member robots read the leader's task instruction regularly. Therefore, the team can have their task updated according to the user.

Adding/removing a task requires the system to re-organize. Assume that a Searcher robot alone is unable to carry one victim, but two robots can collaboratively do it. Initially the Searcher Robot team is assigned to the task of locating the victims, and another team of robots that specialize in rescuing is expected to come when all the victims are found. However, after the searching task is finished, the user finds that the rescuer robots are unavailable for some reason, and assigns the rescuing task to the Searcher Robots. To simplify the implementation, for this case we assume that no barrier exists in the arena.

A Searcher Robotics team *H*(1, 20) is deployed. To meet the user's new requirement, the Leader has to initiate a re-organization that turns the Hierarchy structure to CarrierTeam, as shown in Figure 10 (A).



(A) Leader's behavior rule **(B) Simulation result**

Figure 10 Re-organizing to a CarrierTeam

The CarrierTeam organization comprises two roles: CarrierLeader and Carrier. After re-organizing, the Leader robot becomes the CarrierLeader, and all Members become Carriers. The CarrierLeader assigns a pair of Carrier robots to rescuer each victim to a certain destination. The details of the behavior rules are omitted for the sake of space. A simulation of the model is shown in Figure 10 (B), where the bond lines denote the traces of robots after the system is assigned with the new objective of sending the victims back.

This case illustrates that given a new task, the system has to re-organize such that the agents move to new castes that have capabilities to fulfill the task.

Two more types of re-organization are identified in [4]: (a) changing the duty assignment in the organization, and (b) changing the knowledge of the organization. The only difference between a change of duty and a change of objective is that the subject of duty is the role of a system, while the subject of an objective is a system. In CAMLE, both the roles in a system and the system itself are modeled by castes. Therefore, there is no need to distinguish objectives from duties. For example, when the Searcher Robotics team is assigned a new objective, the robots re-organize so as to fulfill the new requirement, at the same time each robot takes a new duty.

In CAMLE, knowledge is represented as the attributes of castes and the reasoning about knowledge is modeled by internal actions. Different knowledge structure and different formal systems of knowledge reasoning can be encapsulated into castes, and the changes in such knowledge structure and reasoning capability can then be modeled through dynamic casteship changes.

IV. Conclusion

This paper reports a model-driven approach to the design and verification of AMAS using CAMLE. The simulation technique and tool that supports the verifification of decentralized adaptive AMAS is demonstrated with a systematic case study.

In [11], Weyns et al. identified five key requirements on the specification of self-adaptation capabilities: (1) to specify how the system monitors the environment (i.e., context-awareness); (2) to specify how the system monitors itself (i.e., self-awareness); (3) to specify how the system adapts itself; (4) to specify how the system coordinates monitoring and adaptation in a distributed setting; (5) to support extending and refining primitives for additional concerns and domain-specific concepts. Our case study shows that the CAMLE language and its automated tools meet these requirements and provide a strong support to modeling self-adaption. Agents collaborate through mutual observation. By observing the environment and its own state, an agent is both context-aware and self-aware. An agent conducts its actions, including adaption activities, as a reaction to the environment following the prescribed behavior rules. The systematic case study demonstrates that CAMLE is capable of modeling a wide range of adaption mechanisms and simulating highly complicated emergent behaviors. No similar work of such case studies has been reported in the literature as far as we know.

One of the future work is to design and implement statistical model checking functionality in the CAMLE environment.

References

[1] D. Weyns, and M. Georgeff: "Self-adaptation using multiagent systems", Software, IEEE, 2010,vol. 27, (1), pp. 86-91

[2] C. Bernon, M.-P. Gleizes, S. Peyruqueou, and G. Picard: "Adelfe: A methodology for adaptive multi-agent systems engineering": Engineering Societies in the Agents World III (Springer, 2003), pp. 156-169

[3] A. Artikis: "Dynamic specification of open agent systems", Journal of Logic and Computation, 2012,vol. 22, (6), pp. 1301-1334

[4] F. Dignum, and V. Dignum: "A formal semantics for agent (re) organization", Journal of Logic and Computation, 2013, pp. ext058

[5] L. Cernuzzi, A. Molesini, A. Omicini, and F. Zambonelli: "Adaptable multi-agent systems: the case of the gaia methodology", International Journal of Software Engineering and Knowledge Engineering, 2011,vol. 21, (04), pp. 491-521

[6] S.A. DeLoach, and J.C. Garcia-Ojeda: "O-MaSE: a customisable approach to designing and building complex, adaptive multi-agent systems", International Journal of Agent-Oriented Software Engineering, 2010,vol. 4, (3), pp. 244-280

[7] D.B. Abeywickrama, N. Bicocchi, and F. Zambonelli: "SOTA: Towards a general model for self-adaptive systems". Proc. IEEE 21st International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2012, pp. 48-53

[8] L. Shan, R. Shen, J. Wang, and H. Zhu: "Caste-centric development of agent-oriented information systems", in Rennard, J.-P. (Ed.): Handbook of Research on Nature Inspired Computing for Economy and Management (Information Science Reference, Idea Group, 2006, September 13, 2006 edn.), pp. 692-707

[9] L. Shan, and H. Zhu: "CAMLE: a caste-centric agent-oriented modelling language and environment", in Choren, R., Garcia, A., Lucena, C., and Romanovsky, A. (Eds.): Software Engineering for Multi-Agent Systems III. LNCS 3390 (Springer-Verlag, 2005), pp. 144 - 161

[10] X. Mao, L. Shan, H. Zhu, and J. Wang: "An Adaptive Casteship Mechanism for Developing Multi-Agent Systems", International Journal of Computer Applications in Technology, 2008,vol. 31, (1/2), pp. 17 - 34

[11] D. Weyns, S. Malek, and J. Andersson: "FORMS: Unifying reference model for formal specification of distributed self-adaptive systems", ACM Transactions on Autonomous and Adaptive Systems (TAAS), 2012,vol. 7, (1)