An Integrated Approach to Developing Self-Adaptive Software in Open Environments [1]

XINJUN MAO[1], MENGGAO DONG[1], LU LIU[2], HUAIMING WANG[1]

[1]*Department of Computer Science and Technology, National University of Defense Technology, Changsha, Hunan, China 410073*

[2]*School of Computing and Mathematics, University of Derby, Derby, Derbyshire, UK*

{mao.xinjun@gmail.com,.l.liu@derby.ac.uk}

Abstract: One of the main challenges of developing self-adaptive systems in open environments comes from uncertain self-adaptation requirements due to the unpredictability of environment changes and its co-existence with well-defined self-adaptation requirements in self-adaptive systems. This paper presents an integrated approach that combines off-line and on-line self-adaptation together in a unified technical framework to support the development and running of such system. We take self-adaptive system as multi-agent organization and propose a novel dynamic binding self-adaptation mechanism inspired from organization metaphors to specify and analyse self-adaptation. A description language SADL is designed to program well-defined self-adaptation logic at design-time and implement off-line self-adaptation. In order to deal with uncertain self-adaptation, a reinforcement learning method is incorporated with the dynamic binding mechanism, which enables software agents to make decisions on self-adaptation at run-time and implement on-line self-adaptation. Our approach provides a unified framework to accommodate off-line and on-line approaches and a general-purpose methodology to develop complex self-adaptive systems in a systematic way. A supported platform called SADE+ is developed and a case is studied to illustrate the proposed approach.

Keywords*:* Self-adaptive, Agent, Role Dynamic Binding, Organization, Learning

1. INTRODUCTION

Growing complexity of systems as well as dynamic operating environment demand software to be self-adaptive in order to make systems versatile, flexible, resilient, dependable, robust, energy-efficient, recoverable, customizable, configurable, or self-optimizing [1][4][5]. These requirements are widespread in software-intensive systems, socio-technical ecosystems or ultra-large scale systems (ULS) [2]. Self-adaptive software can evaluate the changes occurring in itself and/or the situation environment and respond to the changes by adjusting itself to satisfy the design objectives. Obviously, this software is highly context dependent and has a set of characteristics such as situatedness, continuous adjustment, local control, etc., that increasingly diverge from the traditional ones. This impacts dramatically on the way software systems to be engineered [3]. Developing complex Self-adaptive systems necessitates an attitudes adjustment with respect to the intersection of software engineering and disciplines such as sociology, systems engineering, and so on [2][6].

One of the main challenges in developing self-adaptive systems in open environments comes from uncertain self-adaptation requirements due to the unpredictability of environment changes and its co-existence with well-defined self-adaptation requirements in self-adaptive systems. Some requirements for the self-adaptation of self-adaptive system can be pre-defined if the designers are aware at design-time of predictable changes and the necessary self-adaptation behaviour. For example, in a system designed to respond to cyber-attacks, the designers will be aware of existing attack methods and can put in place the corresponding adjustments. However, when a system is situated in an open environment, it is not possible to counter all possible attack methods in advance as new attack vectors are being continuously created. Therefore, it may not be feasible to anticipate all possible environment changes and their respective self-adaptation actions, i.e., we cannot assume all self-adaptations are known in advance [1].

To develop such complex systems, several challenges should be considered from the viewpoint of software engineering. First, it is not possible to completely anticipate various changes and define self-adaptation requirements at design-time. The off-line self-adaptation decision approach based on pre-defined changes and well-defined self-adaptation logic using rules [20] or strategies [19] is

[1] This work is supported by National Nature Science Foundation of China (No. 61070034 and No. 91024030), Program for New Century Excellent Talents in University.

therefore infeasible (e.g., developers may hard-code self-adaptation logic with some programming languages or specific strategy description languages). As a consequence, on-line self-adaptation approaches are necessary enabling the software to establish or update its self-adaptation strategy at run-time. Second, as the co-existence of both well-defined and uncertain self-adaptation requirements, integrated approach that supports both off-line and on-line self-adaptation and unified methodology that supports variety of self-adaptation requirements should be provided, in order to develop complex self-adaptive systems in a systematic way.

Other parties have undertaken significant research work in the area of software engineering for self-adaptive systems [1][7][8]. Much of research has been devoted to developing self-adaptive systems with well-defined self-adaptation requirements using software technologies like ADL[14], middleware or model[12], languages[19][25][20]. There have also been recent attempts to apply AI (i.e. learning) technology to unpredictable environments thereby enabling on-line self-adaptation decisions to be made [26][27]. However, the previous research has rarely considered the requirements of the co-existence of well-defined and uncertain self-adaptation in self-adaptive systems and the systematic engineering issues like high-level self-adaptation abstractions and mechanism, unified framework and development methodology. As a consequence, constructing self-adaptive systems situated in open environment is still an open problem in the literature of software engineering.

In consideration of the above requirements and issues, this paper presents an integrated engineering approach to developing self-adaptive software with both well-defined and uncertain self-adaptation requirements. The remainder of this paper is organized as follows: Section 2 defines the abstract model and running mechanism of self-adaptive systems; Section 3 introduces the integrated approach to developing self-adaptive systems, including unified framework, self-adaptation description language and self-adaptation learning; Section 4 describes the systematics engineering methodology and the supported platform SADE+ including a case study. The comparisons to other related work are made in section 5 and lastly, the conclusions and future work are discussed in section 6.

2. MODEL AND MECHANISM OF SELF-ADAPTIVE SYSTEM

2.1 Organization Model of Self-Adaptive System

Self-adaptive software is inherently interdisciplinary; the combination of disciplines is primarily dependant on the design metaphors being adopted to build a specific self-adaptive software system [1]. In this paper, the self-adaptive system is modelled as multi-agent organization (see Figure 1) based on agent technology and organization metaphor. Here an agent is considered to be an autonomous entity situated within the environment to satisfy the design objectives. The situatedness and autonomy of agent provide fundamental abstractions to investigate self-adaptation. The organization metaphor provides a novel viewpoint to understanding self-adaptation and building self-adaptive software systems.

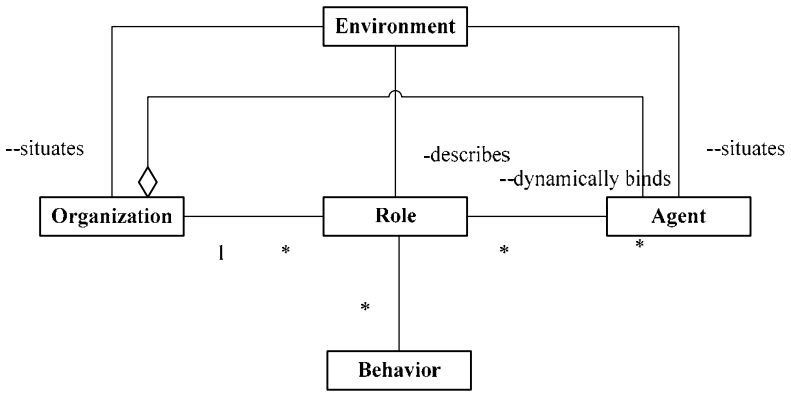


Figure 1. Abstract MAS organization meta-model of self-adaptive system

A self-adaptive system is actually a MAS (Multi-Agent System) organization consisting of various roles and agents in the organization context which defines their environments. Role is the abstract characterization of the behaviour and environment of the agents in the organization. An agent can play multiple roles and a role can be played by multiple agents. Playing a role means that the agent

obtains the behaviour and environment features defined in the role. The environment of an agent or organization is the set of external elements that interact with the agent or organization. Changes in the environment will typically impact on the elements, structure and behaviour of the organization or agent and will therefore result in their adaptive adjustments. In many MAS organizations, the environment of the agent or organization is open, uncontrollable and unpredictable.

2.2 Dynamic Binding Self-adaptation Mechanism

According to the abstract organization model of self-adaptive systems, an agent in the organization can dynamically adjust its roles to adapt to the changes of environment or itself. Such an adjustment can be performed by executing four atomic self-adaptation operations on role: "join", "quit", "suspend" and "resume". When the roles of the agent are adjusted, the structures, behaviours and states of the agent will also be changed. This method of accomplishing self-adaptation is known as dynamic binding mechanism (see Figure 2).

- **join**. Agent can take action to *join* a role, consequently, it obtains the structure, behaviour and environment features defined in the role, and therefore changes its position in the organization context. When agent *joins* a role, the role is said to be bound to by the agent.
- **quit**. Agent can take action to *quit* a role, consequently, it loses the structure, behaviour and environment features defined in the role, and therefore changes its position in the organization context. When agent *quits* a role, the role is said to be unbound by the agent.

The self-adaptation of an agent in an organization context can also take a form of changing the bound roles' status between either "*active*" or "*inactive*". When a role is *actively* bound to, the structure, behaviour and environment of the role will be functional, i.e., the agent can access the properties in the structure, take actions based on the behaviour, and interact with the environment. Alternatively when a role is *inactively* bound to, the agent can access the structure information defined in the role, e.g., introspecting the bound role or querying the properties of the structure, but it will not govern the agent's behaviour nor influence any interaction with the environment.

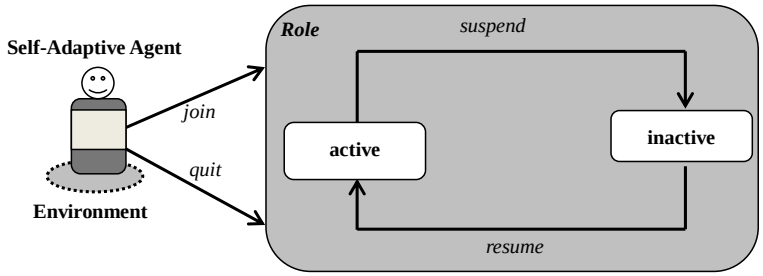


Figure 2. Self-adaptation mechanism of role dynamic binding

- **suspend**. An agent can take action to *suspend* a role; consequently it changes the status of the bound role from "*active*" to "*inactive*".
- **resume**. An agent can take action to *resume* a role, consequently it change the status of the bound role from "*inactive*" to "*active*".

If agents in self-adaptive organization have the capabilities of dynamically binding to roles, we call such agents as self-adaptive agent (abbreviated as SA). A formal definition and reasoning of dynamic binding mechanism can be found in [19] .

3. INTEGRATED APPROACH FOR SELF-ADAPTIVE SYSTEM

3.1 A Unified Software Development Framework

As the degree of self-adaptation uncertainty is variable, it is imperative for software engineering to develop improved models that incorporate various techniques in solving the practical problems of automatic adaptive systems [1]. In order to support the development and running of self-adaptive systems with both well-defined and uncertain self-adaptation requirements, we propose a unified software development framework (see Figure 3).

In our framework, a program of self-adaptive MAS organization consists of three parts: (1) a role class module, (2) a self-adaptive agent module and (3) a learner and/or a self-adaptation strategy module. The role is the elementary module unit that encapsulates the environment, services, properties and behaviour of the system. The self-adaptive agent module encapsulates the dynamic binding

mechanism and important properties and loads the self-adaptive strategy and learner modules; guiding by the self-adaptive strategy and the learning result, the self-adaptive agent can dynamically bind to roles based on internal state or situated environment change.

This framework provides two different methods to deal with the well-defined and uncertain self-adaptation respectively and support the off-line and on-line self-adaptation decisions.

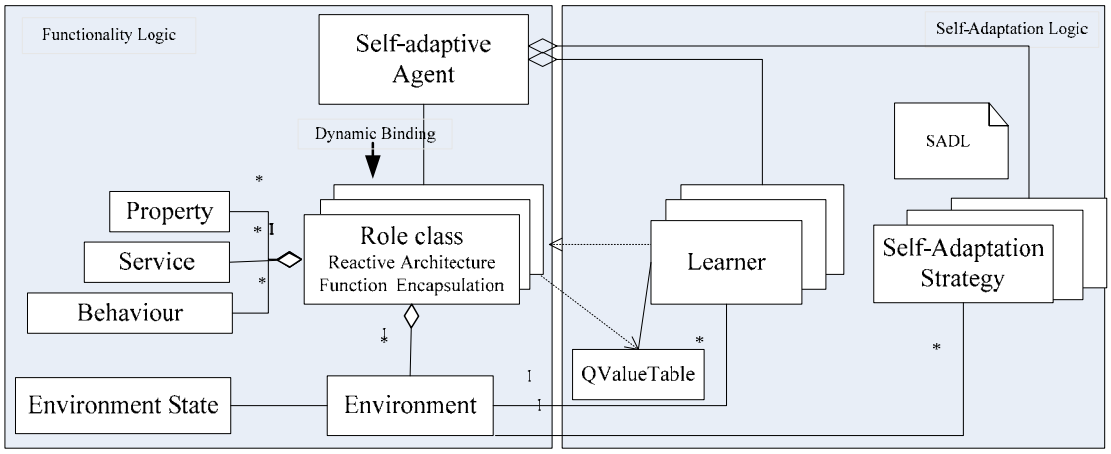


Figure 3. A unified framework to develop self-adaptive systems

If the required changes to the agents can be anticipated and their self-adaptation requirements can be well-defined at design-time, the strategies of self-adaptation should be explicitly programmed with SADL language that we provide. In this scenario, it is actually the software developers that are responsible for defining the self-adaptation logic and making self-adaptation decisions at design-time. The self-adaptation strategy will be interpreted and executed by the supported platform SADE+ that we have developed. If the self-adaptation requirements placed on the agent change from the original scope, the self-adaptation strategy will need to be modified, re-programed and re-compiled. Obviously, this approach is off-line self-adaptation.

When the openness of the situated environment prevents the changes and consequent self-adaptation requirements of agents from being fully anticipated at design-time, an on-line self-adaptation decision process must be adopted. In our approach, on-line self-adaptation is performed by integrating dynamic binding mechanisms and reinforcement learning with each other. This enables the software agent itself to establish the relationship between the observed environment changes and the applied self-adaptive adjustment, thereby determining the response that should be made during run-time. Each self-adaptive agent may contain one or more self-adaptation learners depending on the type and number of the observed changes. The learning results are saved as a knowledge table (e.g., QValueTable). In contrast to the self-adaptation strategy specified by developers with SADL, the knowledge table is dynamically modified and is interpreted by self-adaptive agent to guide the self-adaptation behaviour during run-time.

The proposed framework has the following characteristics. First, it separates the functionality logic from self-adaptation logic of complex self-adaptive system. Each logic is encapsulated in different component (e.g., role class, self-adaptation strategy) based on related technologies (e.g., self-adaptive agent, dynamic binding mechanism, SADL and learning). This is intended as a means to ensure more granular and focus on the different aspects of complex self-adaptive software in open environments, and therefore simplify initial development and subsequent maintenance. Second, off-line and on-line self-adaptations are seamlessly integrated into the framework in a unified way. This integration is based on the common dynamic binding self-adaptation mechanism and software architecture. It provides a flexible way to construct and adapt self-adaptive software according to the uncertainty degree of self-adaptation requirements. Moreover, developers can inherit a number of reusable software package provided by the framework and the corresponding platform SADE+ to improve the development quality and efficiency.

3.2 Self-adaptation Description Language SADL and Off-line Self-adaptation

If the self-adaptation requirements of software agent can be well-defined at design-time,

developers can utilize the off-line self-adaptation design approach and explicitly describe the self-adaptation logic with SADL to develop a self-adaptive agent. SADL is a self-adaptation description language that we present as a means to specify an agent's adaptive strategy. Each self-adaptive strategy can be used by only one agent and each agent can have one or more adaptive strategies in order to satisfy the requirement evolution.

A SADL program consists of a number of declarations and adaptive strategies. Each adaptive strategy consists of a number of adaptive rules that define how an agent adapts to the change in environment. The rules are in the format "WHEN (*EventExp*) IF (*StateExp*) { (*AdaptiveAction*())+ }". Where *EventExp* is the description of environment event, each of which has its name and parameters, *StateExp* is the state expression that describes the state of the self-adaptive agent and *AdaptiveAction* describes how an agent adapts to environment changes by taking actions to adjust its structure and behaviour. An explanation of this statement is given that when the specified internal or environment event occurs, and if agent has the specified state, then agent should take the adaptive actions to adjust its structure and behaviour. Figure 4 depicts parts of SADL syntax definition specified with EBNF.

<*Program*> ::= ("**package**" <*PackageName*> ";")?
("**import**" ((<*PackageName*> "." "*") | ((<*PackageName*> ".")? <*RoleClassName*>)) ";")*
"**use**" ((<*PackageName*> ".")? <*AgentClassName*> ";")
"**program**" <*ProgramName*> "{" (<*InitStrategy*>)? (<*AdaptiveStrategy*>)+ "}" <*EOF*>
<*InitStrategy*> ::= "**InitStrategy**" "{" <*AdaptationStatement*> "}"
<*AdaptiveStrategy*> ::= "**strategy**" <*StrategyName*> "{" (<*AdaptiveRule*>)+ "}"
<*AdaptiveRule*> ::= "**when**" "(" <*EventExpression*> ")" "{" (<*IfStatement*> | <*AdaptationStatement*>)+ "}"
<*InternalEvent*>::= "**INTERNAL_EVENT**"
"(" <*InternalEventName*> "," (<*RoleName*> | "**self**") ("," <*ValueExpression*>)* ")"
<*ExternalEvent*> ::= <*AgentLifeCycleEvent*> | <*AgentAdaptationEvent*> | <*AgentStateChangeEvent*>
| <*AgentBehaviourEvent*> | <*TopicEvent*> | <*ServiceEvent*> | <*SelfDefinedEvent*>
<*AdaptationStatement*> ::= <*CoursegrainAdaptation*> | <*FinegrainAdaptation*>
<*CoursegrainAdaptation*> ::= (<*AdaptiveAction*> '(' <*RoleIdentifier*> (',' <*Expression*>)* ')' ';')+
<*FinegrainAdaptation*> ::= '(' <*RoleIdentifier*> ',' <*BehaviourIdentifier*> (',' <*Expression*>)* ')' ';'
<*AdaptiveAction*> ::= ("**join**" | "**quit**" | "**suspend**" | "**resume**")

Figure 4. Part of SADL Syntax.

- **Declaration:** SADL program should explicitly declare (1) package path of the strategy; (2) the various roles that are imported and used in the strategy; (3) the agent that uses the strategy; and (4) program name.
- **Strategy body:** Each adaptive strategy consists of a number of adaptive rules that define how an agent adapts to the changes based on the dynamic binding mechanism. There is an initial strategy that will be uploaded when the agent is created.
- **Environment and its Change:** The environment of a self-adaptive agent is the set of events that are related with that agent. The change in environment is defined as the occurrence of environment events. The environment of an agent is explicitly defined in the role class that it plays so that the changes of environment can be observed. There are several kinds of environment events predefined in SADL, e.g., Agent Event, Role Event, Topic event, Service Event, etc.
- **Adaptation operations:** There are two kinds of self-adaptations: coarse-grain self-adaptation by adjusting roles of an agent and fine-grain one by adjusting behaviors of an agent. The coarse-grain self-adaptation includes four atomic self-adaptation actions, i.e., *join*, *quit*, *suspend* and *resume*.

The SADL program will be compiled and merged with functionality logic based on the SADE+ platform that we have developed (see section 4).

3.3 Self-Adaptation Learning and On-line Self-adaptation

Self-adaptation requirements of software agents in open environments are often uncertain due to the unpredictability of environment changes. For this type of self-adaptive agents, the off-line approach

cannot be applied because it is not possible for developers to explicitly define self-adaptation logic at design-time. In order to resolve the problem, we present an on-line self-adaptation decision approach that integrates dynamic binding self-adaptation mechanisms and reinforcement learning (see Figure 5). Each self-adaptive element has one or more learning processes. The learners obtain the knowledge that the agents use to select appropriate self-adaptation operations that make suitable on-line decisions in response to unpredictable changes. Therefore, the decision on self-adaptation is made by a software agent during run-time.

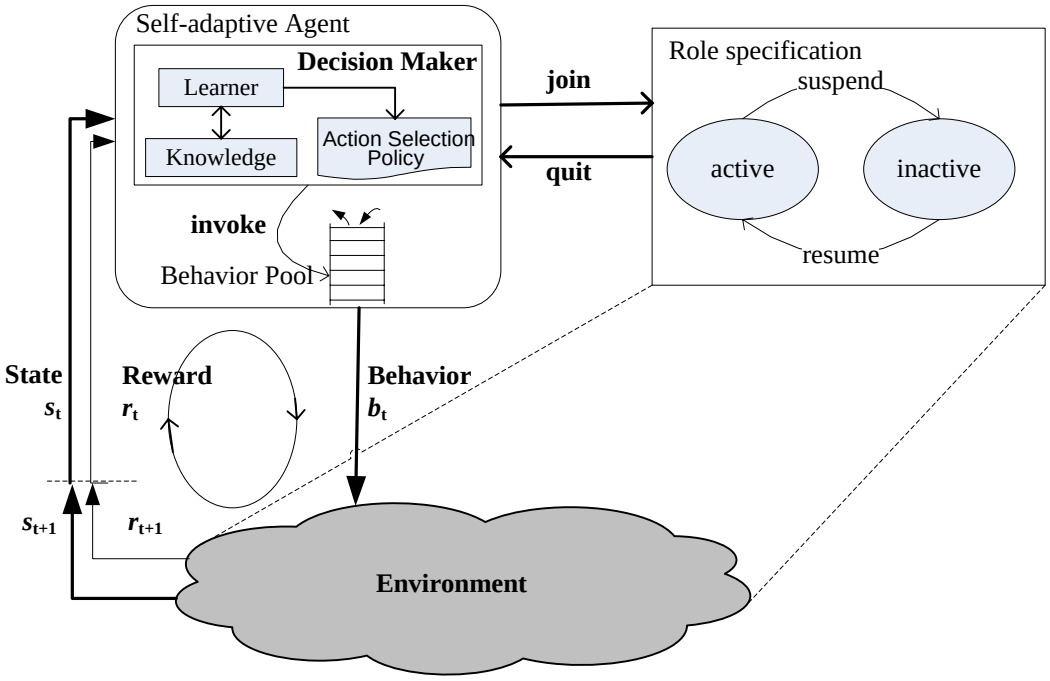


Figure 5. On-line self-adaptation decision approach based on dynamic binding mechanism and reinforcement learning

The process of the agent adapting to the situated environment is similar to the interaction between an agent and its environment; the agent monitors the state of the environment and observes the reward, from the environment after it performs a behaviour. The agent then selects a new behaviour (b_t) based on the current state (s_t) and current reward (r_t). After performing b_t, the environment transits into the next state (s_{t+1}) and produces a new reward (r_{t+1}). This continuous interaction is the basis to realise the self-adaptation. Through interacting with the environment, an agent can learn the knowledge of the mapping between situation and adaptation action, to permit the system to make dynamic and autonomous decisions. The learner is used to implement the learning functionality according to a particular learning algorithm (e.g. Q algorithm); the action selection policy may be the "ε-greedy" selection strategy to choose an action from the action set. The learning result can be saved in various forms, one of which is knowledge table that depicts the values of the actions on states.

The self-adaptation learning algorithm for role dynamic binding to support on-line self-adaptation (see Figure 6) includes four steps. The input of this algorithm is the state of the environment and the self-adaptive agent, the outputs are two Q-tables used to save the learning results when agent binds to different roles. The assumption made in the algorithm is that the agent has bound the role r_1.

First, a Q-table should be created and the Q-values initialized. Second, the current state should be observed and collected; this is regarded as the initial state of learning. Third, the learning loop should be performed. The learning algorithm about behavior (in steps a~c and i~k) and binding roles (in steps a and d~k) are included in this cycle. The execution of the chosen binding actions can make the self-adaptive agent change its bound roles, such as the bound role is changed to r_2 from r_1. At the same time, the different learner and Q-table related to r_2 will be loaded. After that, the agent should learn the execution of the behavior of r_2 according to the learning algorithm about the behavior. When the execution of the behaviors of r_2 finishes, the whole cost (e.g. moving steps or consuming energy) which is used to achieve the objective or implement the task will be calculated. Based on the calculated cost and the reward function, the reward of bound role r_2 can be obtained.

It should be noted that the execution of coarse-grain self-adaptation operations will not affect environment directly. Typically, the environment is influenced only when the behaviours owned by the role have been executed. Hence, binding roles operations have the delayed reward problem, which is different from the normal Q algorithm in traditional reinforcement learning. In the learning phase, the agent accumulates the experiences by updating $Q(s_t,a_t)$ with the equation: $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha[rd + \gamma \max_{a'} Q(s',a')]$, where "$(1-\alpha)Q(s,a)$" is the value that the agent already knows and its weight for accumulation and "$\alpha[rd + \gamma \max_{a'} Q(s',a')]$" is the current reward and the expectation of the greedy action of the newly observed state. α ($0 \le \alpha \le 1$) is a factor that controls the weight and γ ($0 < \gamma \le 1$) is a factor that controls the weight of the greedy action. Finally, two Q-tables will be returned (see Table 1), with which agent can select appropriate operations in different states.

Algorithm: BindingActionLearningAlgorithm
Assumption: Self-adaptive Agent has bound to role r_1
Input: The state of the environment or self-adaptive Agent
Output: The QValueTable(r_1), QValueTable(r_2)
Method:

1. For each pair (s, a), initialize the table entry Q(s, a), where Q(s, a) ∈ QValueTable (r_1);
2. Observe the current state, s;
3. Do forever until s is terminal:
   a) Select an operation a (a may be a binding action, such as *join*, *quit*, *suspend or resume*, or the behavior of the bound role, such as moveRight, etc.);
      If (a is behavior) then do (b ~ c, i ~ k) else do (d ~ k)
   b) Execute behavior a following an exploration strategy (such as ε-greedy);
   c) Receive the immediate reward ***rd***;
   d) Execute a (such as join(r_2)), change the binding role;
   e) Change the loading learner, initialize the parameters of the new learner for r_2;
   f) Start learning according to BehaviorLearningAlgorithm. The learning result is saved in QValueTable(r_2);
   g) Observe or calculate the cost for implementing the task of r2, and then get the reward of binding action (***rd***) according to the reward function about binding to r_2;
   h) Change the loading learner to the one for r_1, the learning about r_1 continues;
   i) Observe the new state, s';
   j) Update the table entry for Q(s, a) as follows:
      $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha[rd + \gamma \max_{a'} Q(s',a')]$
   k) Set s to s'.
4. Return QValueTable(r_1), QValueTable(r_2)

Figure 6. Learning algorithm for on-line self-adaptation

Table 1. Q-table related to Role *r1*

	s_1	s_2	⋯	s_n
$r_1.b_1$	$Q(s_1, r_1.b_1)$	$Q(s_2, r_1.b_1)$	⋯	$Q(s_n, r_1.b_1)$
⋯	⋯	⋯	⋯	⋯
$r_1.b_m$	$Q(s_1, r_1.b_m)$	$Q(s_2, r_1.b_m)$	⋯	$Q(s_n, r_1.b_m)$
$join(r_2)$	$Q(s_1, join(r_2))$	$Q(s_2, join(r_2))$	⋯	$Q(s_n, join(r_2))$

4. SOFTWARE DEVELOPMENT AND PLATFORM SUPPORTS

4.1 Supported Platform SADE+

In order to support the development and running of self-adaptive MAS, we have developed a CASE environment called SADE+ (Self-Adaptation Development Environment). Figure 7 depicts the platform framework which is based on Jade and consists of three levels: development level, running level and infrastructure level.

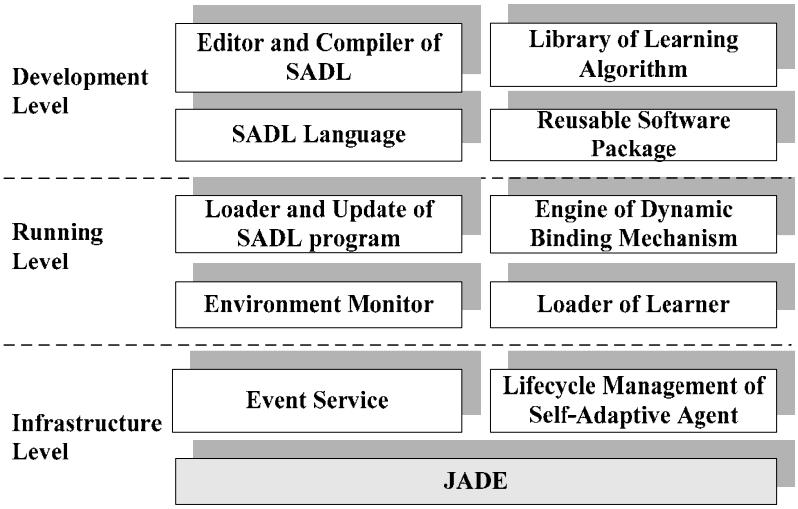


Figure 7. Platform Framework of SADE+

The development level provides the SADL language and library of self-adaptation learning algorithms to program on-line and off-line self-adaptation logic, and the reusable software package to program functionality logic. It also provides an editor and compiler for SADL and integrates them with the Eclipse development environment (see Figure 8). The software package encapsulates and implements basic capabilities of the self-adaptive agent, role, learner, environment, etc. The self-adaptive agent model in our approach is an extension to the reactive architecture of the software agent and supports the self-adaptive decision of the dynamic binding mechanism. All elements of the language, components and tools will assist developers to construct high-quality self-adaptive software more efficiently. The running level provides the loader and update for the SADL program, the engine for the dynamic binding mechanism and the environment monitor. It also provides the loader for the learner to support the deployment as well as running of self-adaptive software. The infrastructure level provides the event service and lifecycle management of the self-adaptive agents.

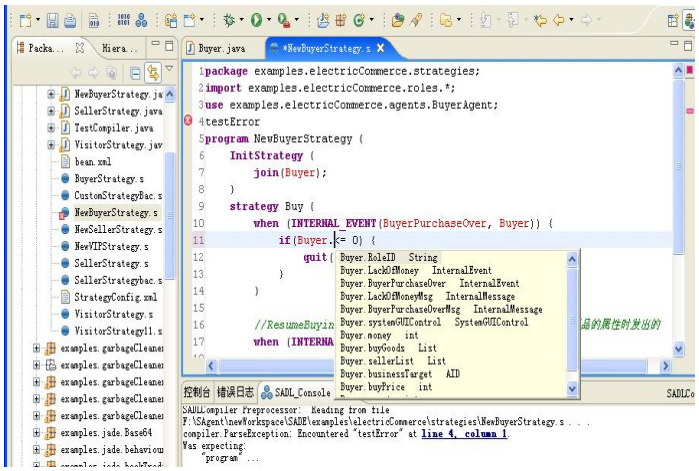


Figure 8. Development Environment of SADE+

Figure 9 illustrates the development and running of self-adaptive software based on SADE+. Developers design and implement the self-adaptation logic and functionality logic of the self-adaptive

software with SADL and Java language respectively. SADL complier is responsible for compiling the SADL program to Java code. MAS codes may inherit the reusable software package and reuse the learning algorithms that are provided by SADE+. All of the Java programs will be further complied into Java byte code, which is executed in the SADE+ running environment. The engine of SADE+ is responsible for integrating the self-adaptation logic and functionality logic at run-time.

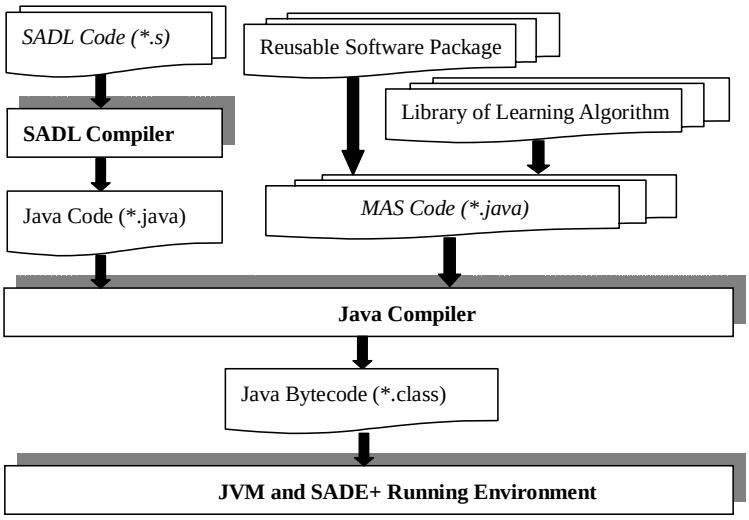


Figure 9. The compile and execution of self-adaptive software in SADE+

4.2 4.2 Unified Development Process and Case Study

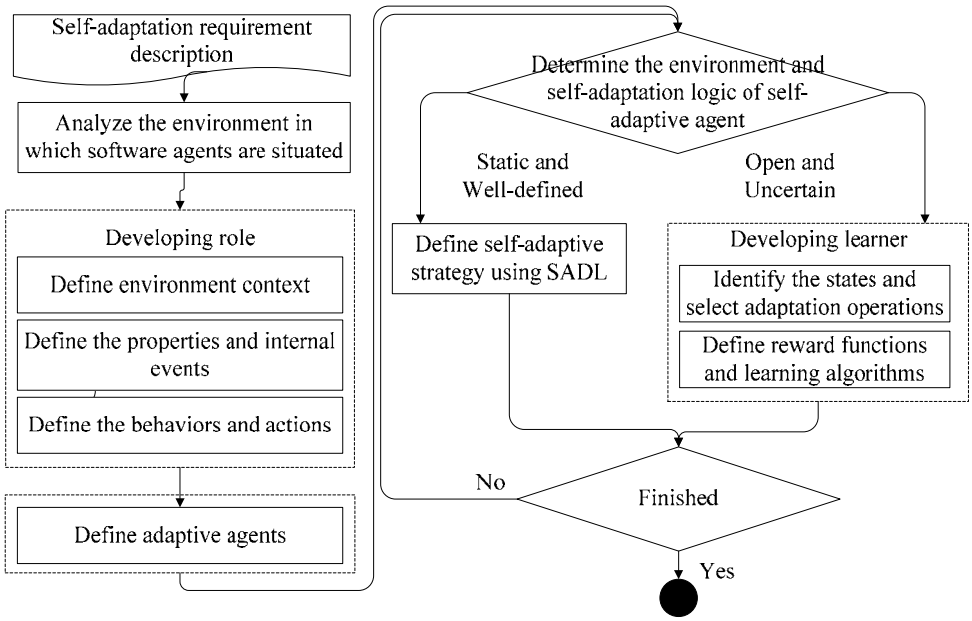


Figure 10. The unified development process of self-adaptive software

In this section, we propose a unified development process and study a case to illustrate the use of our approach to systematically develop self-adaptive systems. The case is based on a hunting multi-agent system in an open environment. The agents in the sample are either hunter agents or prey agents. Each agent can move and has limited energy. The responsibility of a hunter agent is to catch prey agents which move randomly. When a hunter agent lacks energy, it should move to the supply station to get energy. The environment of system is complex and variable, including terrains consisting of sand, mud, stones, etc., each terrain causing different quantities of energy to be consumed. The environment of the agents is uncertain, and changes of the environment are dynamic. For example, an

unexpected or unanticipated change in the environment may be generated if the solid lands become covered by water and become muddy lands. To simplify the design and implementation, the following assumptions are made. First, the hunter agents and prey agents are situated and move around in a 10×10 grid. Second, obstacles in the environment are generated automatically and randomly. Third, in each step, an agent has four possible actions to choose from {*moving up*, *moving down*, *moving left*, *moving right*}, and the hunter agents can sense the moving directions of the prey agents.

First, developers should identify and design the roles of the system by inheriting abstract class Role from the reusable software package provided by SADE+. The properties, behaviours and internal events should be explicitly defined. For example, two roles of "Catcher" and "Supplier" can be identified in the case. The catcher role encapsulates a number of behaviours (e.g., moving, sensing a prey agent in the environment), properties (e.g., the catchers remaining energy and current position) and environment (e.g., the position and moving direction of prey, terrain type). The supplier role encapsulates moving behaviour to search for the supply station. Second, developers should design and implement various agents of the system. If the agent is assigned with self-adaptation capability, it should inherit abstract class *SAgent* from the reusable software package provided by SADE+. In the example, there are at least two kinds of agents: *HunterAgent* and *PreyAgent*. *HunterAgent* is a self-adaptive agent and is designed to catch prey. The program of *HunterAgent* should explicitly declare the class path of the roles that the agent will join and the learner that the agent will load. When a *HunterAgent* is created, it can obtain the capabilities and behaviours to catch a prey agent by joining the Catcher role.

```
package examples.HuntingMAS.strategies;
……
program NormalHunterStrategy { //program name
    strategy {//Scenario: when energy is to be consumed
        when (INTERNAL_EVENT(Energy_Consumed, Catcher)) {
            if (Env.type == Muddy) {
                suspend(Catcher);
                join(Supplier);
            }
        }
        //Scenario: when energy is supplied
        when (INTERNAL_EVENT(Energy_Enough, Supplier)) {
            if (CurPosition == Supply_Station) {
                quit(Supplier);
                resume(Catcher);
            }
        } ……
    }
}
```

Figure 11. The self-adaptation strategy specified with SADL

■ **Development of self-adaptive agents with well-defined self-adaptation requirements**

Some hunting agents in the sample may be assigned limited responsibilities. Changes of the environment can be expected and the self-adaptation requirements can be pre-defined. For example, there is a definition of *NormalHunter* who are designed to catch prey agents in specific and well-defined environments, e.g., sand, flat ground, etc. For this type of agent, the off-line approach and SADL can be used. The following is the self-adaptation strategy for this agent type as specified with SADL (see Figure 11).

■ **Development of self-adaptive agents with uncertain self-adaptation requirements**

Some hunting agents in the example may be assigned powerful self-adaptation capability to deal with an unpredictable environment and uncertain self-adaptation requirements. For example, there is a kind of *PowerfulHunter* who are designed to catch prey agents in open and unpredictable environments, where new or maybe unknown environment events can occur, e.g., new obstacles. For this agent type the on-line approach should be adopted.

Developers should design and implement the learners that support on-line decisions of the

self-adaptive agents. Two learners of *"CatchLearner"* and *"SupplyLearner"* can be identified and designed in the example. The *CatchLearner* is responsible for learning knowledge about catching a prey agent, such as how to move when encountering unanticipated obstacles or sensing the moving direction changes of the prey agents. The *SupplyLearner* is responsible for learning how to go to the supply station with the least energy consumption. When designing learners, four sub-steps should be performed.

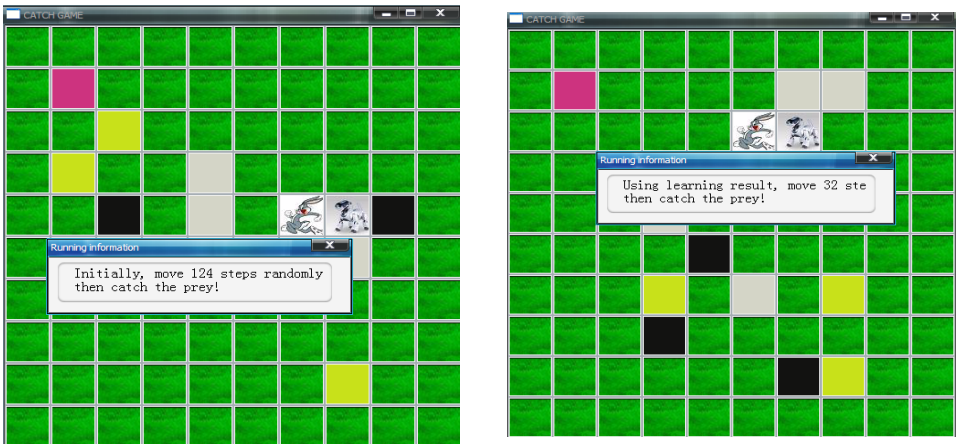
(1) Identify interested states of the environment or agent: The number of states may be very large in many complex systems. However, only some of the states will influence the learning process, others will have no effect. Therefore, it is very important for developers to select appropriately and identify enough states which are both meaningful and effective for learning. In this case, the interested states for learning include the position of the hunter agent, the moving direction of the prey agent, the information about obstacles around the hunter agent and the current hunter energy level.

(2) Choose operations of the learner: A learner concerns the knowledge about which action should be chosen in a specified situation. In our approach, the actions that the learner can choose include two types: the coarse-grain self-adaptation actions and the fine-grain ones. Hence, the *CatchLearner* can choose the actions for adjusting roles (e.g., join, quit) and the behaviour defined in roles such as moving left, right, up and down.

(3) Define reward function: It is very important for a learner to set a proper reward value for each action. In this case, we design the reward rules for the hunter agent moving as follows. For example, it will get -1 when it moves one step without catching the prey agent; it will get different negative values when it hits on or passes different obstacles, for example it will get -2 when it moves one step in sand and -3 when it moves one step in mud; and it will get 100 when it catches the prey agent. For the reward of binding a role,

$$reward = \begin{cases} 50/remainEnergy, if\ remainEnergy \le 50. \\ -100, if\ remainEnergy > 50\ \ or\ moving\ to\ supplystation\ unsuccessfully. \end{cases}$$

(4) Design learning algorithm: The Q-learning algorithm adopted in this case is as shown in Section 3. *LearningAgent* owns an instance of Learner, and it can change the type of the instance to *SupplyLearner* by downcast. *SupplyLearner* is responsible for learning the knowledge about how to go to supply station with the least energy consumption. The learning results are saved in a knowledge table which can be used by the Supplier to guide its moving actions. We have successfully developed the software prototype (see Figure 12) for the example based on the above approach and SADE+.



(a) Catching scenario at the early stage of learning.

(b) Catching scenario at the late stage of learning.

Figure 12. Snapshot of the case study

5. RELATED WORKS

In recent years, software engineering for self-adaptive system has become an active area of research. Substantial research has been undertaken and significant progress has been made [1][7] covering the lifecycle of self-adaptive systems and for various purposes such as QoS[28], monitoring[29] . Typically, previously unrelated disciplines (e.g., control theory and artificial

intelligence [8]) are combined and utilised contrasting computational paradigms (e.g., context-oriented programming, declarative programming, etc. [13]) to support the development and operation of self-adaptive software systems.

Most of the existing research implicitly assumes that changes related to self-adaptation can be anticipated and consequently the requirements on self-adaptation can be precisely defined at the design-time. Therefore, they provide various tools (e.g., rule[20], strategy and language[19][21]) to explicitly define the self-adaptation at various levels such as software architecture or component [15][16]. These technologies are effective for the self-adaptive systems whose environments are genuinely predictable and allow the self-adaptation requirement to be specified beforehand. For example, the concerned changes and how to adjust at architecture level can be formally defined with some ADL language like [14]. However, when the environment in which self-adaptive systems (e.g., ULS [2]) are situated is unpredictable and uncertain, the self-adaptation decision should be shifted from design-time to run-time. It has been stated in [1] that the main challenge of software engineering for self-adaptive systems results from the dynamics of adaptation that requires the well proven principles and techniques valid for standard software engineering to be questioned and new solutions to be considered.

In order to solve the issues coming from the self-adaptive systems in an open environment, several attempts have been made. One is to introduce the learning algorithms into the self-adaptive component (e.g., software agent) to support the dynamic action selection [27] or planning [26]. Adaptive learning could be useful and can act as a strategy to achieve self-adaptivity [8]. However, the researches on this topic often focus on the uncertainty aspects of self-adaptation and are involved in the learning algorithms details.

Recently, agent technology and organization metaphors have been utilized as fundamental abstractions to model, analyse and design self-adaptive systems. Several meta-models, modelling languages and methodologies are proposed. Juan [10] presents a meta-model based on agent, role, service and protocol concepts for intelligent adaptive MAS in open environments. Some organization abstractions such as organization rule, norm, etc., are introduced into agent-oriented methodology (e.g., Gaia [11], O-MaSE[17], ADELFE [18]) to support the analysis and design of self-adaptive systems. Significantly, role playing and allocation are designed as mechanisms to model and analyse self-adaptation [23][24]. Many of these approaches are proposed to support the high level analysis and design of self-adaptive systems, and seemingly the implementation and running are seldom considered. One of the related works [26] captures the role dynamics of agents in open systems in terms of four operation "*enactment*", "*deactment*", "*activate*", and "*deactivate*". However, the semantics of the operations on the role are different from the operations "*join*", "quit", "*suspend*" and "*resume*" defined in this paper. Moreover, 3APL assumes that at any moment only one role can be active and that all roles other than the enacted one are inactive. [30] argues the necessity to unify the reference model and presents a formal one called FORMS that supports the description and reasoning of self-adaptive systems.

6. CONCLUSIONS AND FUTURE WORK

The main challenges of developing self-adaptive systems in open environments come from the unpredictability of environment changes, uncertainty of self-adaptation requirements, and the co-existence with well-defined self-adaptation requirements. Obviously, to develop such complex systems needs high-level abstractions and mechanism, unified technical framework and systematics methodology to manage and control complexity.

This paper presents an integrated approach that combines off-line and on-line self-adaptation together to deal with the above challenges. Our contributions are threefold. First, we propose a novel dynamic binding self-adaptation mechanism inspired from organization metaphors. Such a mechanism provides high-level abstractions and models to investigate and achieve self-adaptation independent of implementation technologies and platforms. Second, we propose a unified framework that integrates off-line self-adaptation and on-line self-adaptation together and supports developers to implement self-adaptive systems with well-defined and uncertain self-adaptation requirements. In order to program well-defined self-adaptation logic at design-time and implement off-line self-adaptation, SADL based on the dynamic binding mechanism is proposed. Our approach separates the functional behaviours from adaptation behaviours in order to simplify the development and maintenance of self-adaptive systems. Reinforcement learning methods are incorporated with the dynamic binding mechanism to enable software agents to make decisions on self-adaptation at run-time and implement

on-line self-adaptation. Third, we have presented a unified development process and platform SADE+ to support the implementation of self-adaptive systems in open environments in a systematic and efficient way using our proposed approach. Based on the proposed approach, we have studied several cases and successfully developed a number of software prototypes of self-adaptive systems such as adaptive garbage clearing MAS, self-adaptive electronic commerce, flexible conference management system, etc.

Self-adaptation can be performed at different levels such as data, component, architecture and even organization. This paper only considers the self-adaptation at the agent level. Our research in the next step will involve the self-adaptation at the architecture and organization levels. Mechanism design for understanding and achieving the self-adaptation at these levels should be considered based on the intersection with related disciplines such as sociology, organization, etc. Moreover, technologies that integrate the self-adaptation mechanism and closed feedback loop should be developed. Another attempt is to develop a systematic methodology covering the analysis, design, implementation and deployment of self-adaptive systems with unified abstractions and meta-models. Especially we intend to bridge the gap between the high-level analysis and design models with the low-level implementation model with model driven development techniques.

REFERENCES

[1]. Cheng, B., et al,. Software Engineering for Self-Adaptive Systems: A Research Roadmap. In Proc. Of Software Engineering for Self-Adaptive Systems, LNCS 5525, 2009, 1-26.
[2]. Linda Northrop, Ultra-Large-Scale Systems: The Software Challenge of the Future, Software Engineering Institute, Carnegie Mellon University, Software Engineering Institute, 2006.
[3]. Franco Zambonelli, H. Van Dyke Parunak, Towards a Paradigm Change in Computer Science and Software Engineering: A Synthesis, The Knowledge Engineering Review, 18(4): 329-342, 2003.
[4]. Markus C. Huebscher and Julie A. McCann, A survey of Autonomic Computing - Degrees, Models, and Applications, 2008, ACM Computing Surveys, 40(3): 1-28.
[5]. Lv Jiang, Xiaoxin Ma, Xianping Tao, Feng Xu, Hao Wu, Research and Development of Internetware. Science in China Series. E Information Sciences, 2006, 36(10):1037-1080.
[6]. Barry Boehm, A View of 20th and 21st Century Software Engineering, Proc. Of ICSE, 12-29, 2006.
[7]. Thomas Kuhne, Current Trends for Self-* Systems - Research Roadmap for Self-Adaptive Systems, Seminar SS Report, 2011.
[8]. Mazeiar Salehie and Ladan Tahvildari, Self-adaptive software: landscape and research challenges. ACM Transactions on Autonomous and Adaptive Systems, 4(2):1-42, 2009.
[9]. Paola Inverardi and Massimo Tivoli, The Future of Software: Adaptation and Dependability, A. De Lucia and F. Ferrucci (Eds.): ISSSE 2006-2008, , LNCS 5413, 1-31, 2009
[10]. Juan, T., Sterling, L. A Meta-model for Intelligent Adaptive Multi-Agent Systems in Open Environments, in Proceedings of AAMAS, 14-18, 2003
[11]. Cernuzzi, L., Zambonelli, F., Dealing with Adaptive Multi-Agent Organizations in the Gaia Methodology. In Proceedings of AOSE, LNCS 3950, 109-123, 2006.
[12]. Giovanna Di Marzo Serugendo, John Fitzgerald, Alexander Romanovsky, and Nicolas Guelfi, MetaSelf - A Framework for Designing and Controlling Self-Adaptive and Self-Organising Systems, Technical Report BBKCS-08-08, School of Computer Science and Information Systems, Birkbeck College, London, UK, 2008.
[13]. Roberto Bruni, Andrea Corradini, Fabio Gadducci, A Conceptual Framework for Adaptation, In: Proceedings of 15th the International Conference on Fundamental Aspects of Software Engineering (FASE'12). Springer, 2012.
[14]. Garlan, D., Cheng, S.W., Huang, A.C. Rainbow: Architecture-based self-adaptation with reusable infrastructure. Computer, 37(10): 46-54, 2004.
[15]. Yoder, J.W., Balaguer F., Johnson, R. Architecture and design of adaptive object models, ACM SIGPLAN Notices, 36(12): 50-60, 2001.
[16]. Lee, S., Oh, J., Lee, E. An Architecture for Multi-agent Based Self-adaptive System in Mobile Environment, in Proc. Of Intelligent Data Engineering and Automated Learning, LNCS 3578, 494-500, 2005.
[17]. Scott A. DeLoach, O-MaSE: A Customizable Approach to Designing and Building Complex, Adaptive Multiagent Systems, International Journal of Agent-Oriented Software Engineering, 4(3):

244-280, 2010.
[18]. Carole Bernon, et.al., ADELFE: a Methodology for Adaptive Multi-Agent Systems Engineering, In Proc. Of Engineering Societies in the Agents World III, LNCS 2577, 70-81, 2003.
[19]. Mao, X.J., Shan, L.J., Zhu, H., Wang, J. The Adaptive Casteship Mechanism for Developing Multi-Agent Systems, International Journal of Computer Applications in Technology, 31(1/2): 17-34, 2008.
[20]. Qianxiang Wang. Towards a Rule Model for Self-adaptive Software, ACM SIGSOFF Software Engineering Notes, 30(1): 1-5, 2005.
[21]. Ji Wang, Rui Shen, and Hong Zhu. Towards agent oriented programming language with caste and scenario mechanisms, in Proc. Of International Joint Conference on Autonomous Agents and Multiagent Systems, ACM Press, 1297-1298, 2005.
[22]. Mckinley, P.K., Sadjadi, S.M., Kasten, E.P., Cheng, B., Composing Adaptive Software. Computer, 37(7): 56-64, 2004.
[23]. Mark Hoogendoorn and Jan Treur, An Adaptive Multi-Agent Organization Model Based on Dynamic Role Allocation, International Journal of Knowledge-Based and Intelligent Engineering Systems, 13(3/4): 119-139, 2009.
[24]. Hilaire, V., Koukam, A., Gruer, P. A Mechanism for Dynamic Role Playing, in Proceedings of International Workshop on Agent Technology, LNAI 2592, 36-48, 2003.
[25]. Dastani, M., Birna van Riemsdijk, M., Hulstijn. Enacting and Deacting Roles in Agent Programming, in Proc. of AOSE, LNCS 3382, 189-204, 2005.
[26]. Kim, D., Park, S. Reinforcement learning-based dynamic adaptation planning method for architecture-based self-managed software. In Proc. Of SEAMS, 76-85, 2009.
[27]. Amoui, M., Salehie, M., Mirarab, S., Tahvildari, L. Adaptive Action Selection in Autonomic Software Using Reinforcement Learning, in Proc. of ICAS, 175-181, 2008.
[28]. Tharindu Patikirikorala, Alan Colman, Jun Han, Liuping Wang, A Multi-model Framework to Implement Self-managing Control Systems for QoS Management, In: Proc. Of SEAMS 2011, 218-227, 2011.
[29]. Teemu Kanstrén, Reijo Savola, Sammy Haddad, Artur Hecker, An Adaptive and Dependable Distributed Monitoring Framework, International Journal on Advances in Security, 4(1): 80-94, 2011.
[30]. Danny Weyns, Sam Malek, Jesper Andersson, FORMS: Unifying Reference Model for Formal Specification of Distributed Self-Adaptive Systems, 2011.