An Integrated Approach to Developing Self-Adaptive Software

XINJUN MAO¹², MENGGAO DONG², LU LIU³ AND HUAIMING WANG²
¹State Key Laboratory of Software Development Environment
Beihang University
Beijing, 100191 P.R. China
²Department of Computer Science and Technology
National University of Defense Technology
Changsha, 410073 P.R. China
³School of Computing and Mathematics
University of Derby
Derby, DE22 1GB UK

One of the main challenges of developing self-adaptive systems in open environment 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 systems. We consider self-adaptive system as a multi-agent organization and propose a novel dynamic binding self-adaptation mechanism inspired from organization metaphors to specify and analyze 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 environments requires software to be self-adaptive in order to make systems more versatile, flexible, resilient, dependable, robust, energy-efficient, recoverable, customizable, configurable, or self-optimizing [1-3]. These requirements are widespread in software-intensive systems, socio-technical ecosystems and ultra-large scale systems (ULS) [4]. Self-adaptive software can evaluate the changes occurring in itself and/or the situated 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, contin-
uous adjustment, local control, etc., that increasingly diverge from the traditional ones. This dramatically impacts on the way software systems to be engineered [5]. Developing complex self-adaptive systems necessitates an attitude change with respect to the intersection of software engineering and disciplines such as sociology, systems engineering, and so on [4, 6].

One of the main challenges in developing self-adaptive systems in open environment comes from uncertain self-adaptation requirements due to the unpredictability of environment changes and its co-existence with well-defined self-adaptation requirements. 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 enumerate 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-adaptive 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 [7] or strategies [8] is 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 to enable the software to establish or update its self-adaptation strategy at runtime. 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, 9, 10]. Much of research has been devoted to developing self-adaptive systems with well-defined self-adaptation requirements using software technologies like ADL [11], middleware or model [12], languages [7, 8, 13]. 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 [14, 15]. However, the previous research has rarely considered 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 an 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 of well-defined and uncertain self-adaptation requirements. The remainder of this paper is organized as follows: Section 2 defines an abstract model and running mechanism of self-adaptive systems; Section 3 introduces an integrated approach to developing self-adaptive systems, including unified framework, self-adaptation description language and self-adaptation
learning; Section 4 describes a systems engineering methodology and the supported platform SADE+ including a case study. The comparisons with 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 dependent on the design metaphors adopted to build a specific self-adaptive system [1]. In this paper, the self-adaptive system is modelled as multi-agent organization (see Fig. 1) based on agent technology and organization metaphor. Here an agent is considered to be an autonomous entity situated in 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 understand self-adaptation and build self-adaptive software systems.

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 is called as dynamic binding mechanism (see Fig. 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 neither govern the agent’s behaviour nor influence any interaction with the environment.

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 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 [8].

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 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 Fig. 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 behaviours of the system. The self-adaptive agent module encapsulates the dynamic binding mechanism and important properties and functionalities such as the output from the self-adaptive strategy and learner modules. With this information, 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.

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-programmed 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) that actually defines the self-adaptation logic. 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.

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 self-adaptive agent. SADL is a self-adaptation description language 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 self-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 self-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. Fig. 4 depicts parts of SADL syntax definition specified with EBNF.

Fig. 4. Part of SADL syntax.

**Declaration** SADL program should explicitly declare (1) package path of the strategy; (2) various roles that are imported and used in the strategy; (3) 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. 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 four atomic self-adaptation actions, i.e., join, quit,
suspend and resume that can be performed by agent to adapt to the changes.

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, we present an on-line self-adaptation decision approach that integrates dynamic binding self-adaptation mechanisms and reinforcement learning (see Fig. 5). Each self-adaptive element has one or more learners. The learners obtain and develop the knowledge that the agents use to select appropriate self-adaptation operations in response to unpredictable changes. Therefore, the decision on self-adaptation is made by a software agent during run-time.

![Fig. 5. On-line approach based on dynamic binding mechanism and reinforcement learning.](image)

The agent monitors the state of the environment and observes the reward from the environment after it performs a behaviour, 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 realize 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 “$\epsilon$-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 Fig. 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 optional operations related to various states include four atomic operations on self-adaptation, i.e., join, quit, suspend and resume. First, a Q-table should be created and the Q-values initialized. Second, the current state should be observed and collected. Third, the learning loop should be performed. Typically, the environment is influenced only when the behaviours owned by the role have been executed. Hence, self-adaptation operations have the delayed reward problem, which is different from the traditional normal Q algorithm. The execution of the chosen self-adaptation actions may cause the self-adaptive agent to adjust its bound roles. In this phase, the agent accumulates the experiences by updating \(Q(s,a)\) with the equation:

\[
Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_a Q(s',a')]
\]

where \((1-\alpha)Q(s,a)\) is the value that the agent already knows and its weight for accumulation and \(\alpha[r + \gamma \max_a Q(s',a')]\) is the current reward and the expectation of the greedy action of the newly observed state. \(\alpha(0 \leq \alpha \leq 1)\) is a factor that controls the weight and \(\gamma(0 < \gamma \leq 1)\) is a factor that controls the weight of the greedy action. Finally, two Q-tables will be returned, 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 \(s\) of the environment and self-adaptive agent

Output: The QValueTable\((r_1)\), QValueTable\((r_2)\)

1. For each pair \((s,a)\), initialize the table entry \(Q(s,a) \in \text{QValueTable}(r_1)\), where \(a\) is self-adaptation action such as join, quit, suspend or resume.
2. Observe the current state \(s\);
3. Do forever:
   a) Select a self-adaptation action \(a\) to execute (e.g., join\((r_2)\)), change the bound roles of agent;
   b) Change the learner and initialize its parameters for new bound role (e.g., \(r_2\));
   c) Learn the behaviour effect of new bound role and save the result in QValueTable\((r_2)\);
   d) Calculate the cost to achieve the task of new bound role, get the reward of the binding action according to the reward function about binding to a new role;
   e) Change the loading learner to the one for \(r_1\), the learning about \(r_1\) continues;
   f) Observe the new state, \(s'\);
   g) Update the table entry for \(Q(s,a)\) as follows:

\[
Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_a Q(s',a')]
\]

d) Set \(s\) to \(s'\).
4. Return \{QValueTable\((r_1)\), QValueTable\((r_2)\)\}

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

4. SOFTWARE DEVELOPMENT AND PLATFORM SUPPORT

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†). Fig. 7 depicts the platform framework which is based on Jade and consists of three levels: development level, running level and infrastructure level.
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 Fig. 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. 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.

4.2 Unified Development Process and Case Study

In this section, we propose a unified development process (see Fig. 9) and study a
case to illustrate our approach. The case is based on a hunting multi-agent system in an open environment. The agents 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 is complex and variable, including terrains consisting of sand, mud, stones, etc., each terrain causing different quantities of energy to be consumed. The changes of the environment are dynamic and uncertain. For example, an unexpected change 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 are generated automatically and randomly. Third, in each step, an agent has four possible moving actions to choose from {up, down, 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. The catcher role encapsulates a number of behaviours (e.g., moving, sensing a prey agent), properties (e.g., the 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.

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 is 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 Fig. 10).

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

Some hunting agents in the sample 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 is designed to catch prey agents in open and unpredictable environments, where new or maybe unknown environment events may 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. 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 learns 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 up 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,

$$
\text{reward} = \begin{cases} 
50 / \text{remainEngery}, & \text{if } \text{remainEngery} \leq 50. \\
-100, & \text{if } \text{remainEnergy} > 50 \text{ or moving to supply station unsuccessfully.}
\end{cases}
$$

(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 Fig. 11) for the example based on the above approach and SADÉ+.

![Fig. 10. Strategy specified with SADL.](image1)

![Fig. 11. Software prototype of the case.](image2)
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, 9] covering the lifecycle of self-adaptive systems. Typically, previously unrelated disciplines (e.g., control theory and artificial intelligence [10]) are combined and utilised contrasting computational paradigms (e.g., context-oriented programming, declarative programming, etc. [16]) 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 [7], strategy and language [8, 17] to explicitly define the self-adaptation at various levels such as software architecture or component [18, 19]. These technologies are only 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 [11]. However, if the environment in which self-adaptive systems (e.g., ULS [4]) 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 [18] or planning [14]. Adaptive learning could be useful and can act as a strategy to achieve self-adaptivity [10]. 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, analyze and design self-adaptive systems. Several meta-models, modelling languages and methodologies are proposed. Juan [20] 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 [21], O-MaSE [22], ADELFE [23]) to support the analysis and design of self-adaptive systems. Significantly, role playing and allocation are designed as mechanisms to model and analyze self-adaptation [24, 25]. Many of these approaches have been proposed to support the high level analysis and design of self-adaptive systems, but the implementation and running seems to be seldom considered.

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. 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 studied. 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 models with model driven development techniques.

REFERENCES


Xinjun Mao (毛新军) Ph.D., Professor and Ph.D. supervisor in National University of Defense Technology in China, member of ACM, IEEE and CCF, His current research interests include software engineering, self-adaptive and self-organizing systems, agent theory and technology, autonomic computing.

Menggao Dong (董孟高) received his MSc and Ph.D. in Computer Science from National University of Defense Technology, China, in 2005 and 2011 respectively. His research interests include agent-oriented software engineering and self-adaptive software.

Lu Liu (刘路) is currently University Reader in Distributed Computing in the University of Derby in UK, He received a Ph.D. degree from Surrey Space Center, the University of Surrey and M.Sc degree from Brunel University. His current research interests include cloud computing, service-oriented computing, and peer-to-peer networking, wireless sensor networks, the Internet of Things.

Huaiming Wang (王怀民) Ph.D., Professor and Ph.D. supervisor in National University of Defense Technology, member of ACM and CCF. His current research interests include distributed computing, software engineering, information security and self-adaptive software.