A Context-Aware Multi-Agent Systems Architecture for Adaptation of Autonomic Systems

Kaiyu Wan, Vasu Alagar

A Context-aware Multi-agent Systems Architecture for Adaptation of Autonomic Systems

Kaiyu Wan[1], and Vasu Alagar[2]

[1] Xi'an Jiaotong-Liverpool University, Suzhou, PRC
[2] Concordia University, Montreal, Canada
kaiy.wan@xtjlu.edu.cn,alagar@cse.concordia.ca

Abstract. An important requirement of autonomic systems is that they self-adapt, both with respect to internal self-healing and with respect to external environmental changes. In order to fulfill this requirement autonomic systems must have awareness abilities, context gathering mechanisms, context-dependent adaptation policies, and the ability to react with respect to each adaptation requirement. In this paper we provide an insight into these issues and propose a multi-agent system that interact faithfully among themselves in order to self adapt with safety and security.

1 Introduction

The autonomic computing system (ACS) concept was initiated at IBM [6], as a solution to reduce the cost of resource management and system administration of large evolving systems. Since its proposal in 2001, several prototype examples of ACS are reported to have been built. These include IBM Tivoli Management Suite [10], SUN Microsystems [11], Hewlett-Packard's Adaptive Enterprise [7], and Microsoft's Dynamic Systems Initiative [9]. We believe that the self-adaptation issue for ACS is still an open and challenging problem. Motivated by this belief we provide a multi-agent system architecture in which context awareness and context-dependence are central to adaptation. The important characteristics of ACS [2, 3] are as follows:

Self-configuring: It automatically adapts to dynamically changing environments.
Self-healing: It detects, diagnoses, and recovers from any damage that occurs during its life cycle.
Self-optimizing: It monitors and tunes its resources optimally.
Self-protecting: It detects and guards itself against damage from accidents and outside attacks.
Among these characteristics, the self-adaptability of ACS is the most effective approach necessary to tackle its immense software management complexity. It is called for during self-configuration, self-healing, and self-protecting scenarios. The system can self-configure when (1) due to its self-monitoring it has *sensed* that some internal updates have become available, or (2) it *perceives* changes to its environment or (3) when *stimulus events* are received from the environment. In the first two cases *anticipatory* adaptation is called for and in the third case a *reactive* adaptation is called for. To enact an anticipatory adaptation, the ACS has to be *context-aware*, and to trigger a reactive adaptation the system must act in a *context-dependent* manner. Anticipatory adaptation is called for

during self-healing as well. The system should monitor its internal contexts which include resources, users, and programs, and include externally perceived contexts for adaptive action when a failure happens. The main goal here is to restore system consistency without disrupting services. Although the adaptive actions can be triggered only after failures, it is important that the system is able to see the sign of failure before it happens. That is the reason we classify this adaptation type as anticipatory. Self-protection requires anticipatory adaptive actions. The system should protect its critical resources from unauthorized use and detect external attacks before damage is caused. So, we conclude that a broad classification of adaptation types can be 'anticipatory' (*before*) and 'reactive' (*reactive*), and both adaptation types involve *context* and *awareness*. Anticipatory adaptation requires *context-awareness* and reactive adaptation requires *context-dependence*.

1.1 Our Contribution

Basic concepts on context and context awareness are discussed in Section 2. Sections 3 and 4 describe the main contribution of this paper. An adaptation logic in a formal setting is given in Section 3.1. In Section 4 we discuss a MAS architecture in which context-aware and context-dependent adaptive actions are possible, and give a detailed agent interaction protocol. The agent collaborations in MAS offer a more effective solution for any large scale autonomic system adaptation.

Traditionally, self-adaptation is realized in a system by adding adaptation logic to the managed system. In our work we embed adaptation in a logic of context. In simple terms, adaptation may be viewed as a sequence of activities "*monitor (M)*, *analyze (A)*, *plan (P)*, and *execute (E)*". Monitoring includes self-monitoring (internal monitoring) and external monitoring. The analysis stage explores the available options for responding to the perceived situations. Often business policies and security policies will constrain the available options. The planning stage puts the options in a certain order of execution, may be in sequential or in parallel, and gives the plan to the execution stage. This sequence $\langle M - A - P - E \rangle$ of activities may have to be performed cyclically, where one cycle may include many smaller inner cycles. In [12] a mechanism to manage the intra-cycle and the inter-cycle coordination for the $\langle M - A - P - E \rangle$ paradigm has been proposed, and is illustrated for a traffic management system. In contrast, our approach is to lift $\langle M - A - P - E \rangle$ cycles to agent collaborations within a multi-agent system (MAS). System adaptation is achieved via agent collaboration. Consequently, the agents are made context-aware and their reactions become context-dependent.

2 Basic Concepts

Adaptation logic will include specific rules as well as meta inference rules. Both types of rules are usually applicable in many different contexts. Context information is both heterogeneous and multi-dimensional. A formal representation of context is essential in order to make agents detect and react to contextual changes. We use the syntax of context and the logic of context introduced by Wan [13]. In Section 2.1 we give a quick review of context definition, representation, and calculation of context expressions. Awareness for

computing systems should mean perception, collection, and a rational use of the information towards a specified goal. The system should not only understand why some information is collected but also be intelligent to use it with a justifiable reason. Information related to internals such as computing resources, computational states, and policies for state change, and information related to its externals such as physical devices, other systems, and humans with which it interacts provide the awareness envelope. In Section 2.2 we discuss the internal monitoring, called *self-awareness* of the system. In Section 2.3 we discuss the external monitoring, called *context-awareness*. Adaptation, regardless of the specific rules binding it, should be based on contextual conditions.

2.1 Context and Situation

Context types are defined in [13] by defining a finite set of dimensions and a tag type for each dimension. Examples of dimensions are $LOCATION$, $TIME$, and WHO (or $NAME$ or $ROLE$). The tag type for $LOCATION$ can be the set of cities or geographical zones, and these are chosen to suit the system need. In general, given the set of dimensions $DIM = \{D_1, D_2, \ldots, D_n\}$, and X_i is the tag type for D_i a context c over these dimensions has the representation $[D_1 : x_1, \ldots, D_n : x_n]$, $x_i \in X_i$. As an example, $[HC : H123456, LOC : NewYork, DATE : 2003/09/30]$ defines a context with dimensions HC (health center), LOC (location), and $DATE$ (calendar date). Many events or situations that might happen in this context. A calculus of context and a toolkit based on it have been extensively used in many context-aware system applications [8, 1]. In the rest of our discussion we will be using the context representation defined here.

A context in itself is not very useful unless it is associated with *events* or *situations* of interest in a system. We formulate a situation as a logical formula involving dimension names and system variables. In general, if α is a logical formula denoting a situation, we write $\mathbf{vc}(c, \alpha)$ to mean that the situation is valid in context c. As an example, the situation $VERYWARM = TEMP > 40 \wedge HUMID > 80$ is true in context $[TEMP : 41, HUMID : 81]$, false in context $[TEMP : 38, HUMID : 85]$, and cannot be resolved in context $[HUMID : 78, WINDSPEED : 35]$. In general, (1) there may exist zero or more contexts in which a situation might be true, and (2) many situations may be valid in a context.

For self adaptation, the system must be able to evaluate situations automatically at different contexts. To enable this automatic evaluation we insist (1) a situation involves only dimension names and system variables, and (2) the situation is formulated as a Conjunctive Normal Form (CNF). The validation is then automated with the following steps:

1. Write the situation as $\alpha = p_1 \wedge p_2 \ldots, \wedge p_k$, where each p_i is a disjunctive clause involving the $DIMENSION$ names and system variables.
2. To evaluate α in a context c, evaluate the p_is in context c until either all of them evaluate to true or a p_i evaluates to false, or a p_i cannot be evaluated. In the first case, the situation α is true in context c. In the second case the situation α is false in context c. In the third case the context or system information is inadequate to evaluate the situation α. The evaluation of each p_i is done as follows:
   - if D is a $DIMENSION$ name in context c, and D occurs in the predicate p_i then the D in p_i is replaced by the tag value associated with D in c,

- substitute the variables in p_i by their respective values, as provided by the system in the environment of evaluation,
- if p_i still has $DIMENSION$ names or variables, then p_i cannot be evaluated in context c; otherwise p_i is now a proposition which evaluates to either true or false.

2.2 Self-awareness

We restrict to the self-awareness behavior related to protecting the critical resources and states of the system, and regulating services. It is necessary to include the following information for modeling self-awareness contexts:

1. *Roles* are defined on the subjects. Each subject must have a role in order to be active in the system.
2. *Resources* are the objects (files such as patient files, policy documents) that are under use. Organizational policies will constrain the access to resources. With each resource is associated a list of roles, indicating for each role the level of access to the resource.
3. *Permission* for each object in the system and for each role is specified. An access control list exists for each resource that specifies explicitly the actions that a role is authorized to perform on the resource, as well as the actions not allowed to perform.
4. *Obligation* is the set of mandatory actions to be performed by the subjects at certain stages.

From the above modeling elements, the context type for self-awareness can now be constructed. We call this set of contexts as *internal contexts (IC)*. Assume that a set of roles is $RC = \{RC_1, \ldots, RC_m\}$. Let $DC = \{DC_1, \ldots, DC_k\}$ be the set of resource categories which are to be protected. We regard RC_i's and DC_js as dimensions. As an example, the context $[RC_1 : u_1, DC_j : \texttt{Salary\_File}]$ might be constructed to indicate that user u_1 plays the role RC_1 while in possession of the resource `Salary_file`. Let PC denote the purpose dimension. Assume that the tag type for PC is the enumerated set $\{\texttt{Technical}, \texttt{Auditing}, \texttt{Research}\}$. An example of a self-awareness context with these three dimensions is $[RC_1 : u_1, DC_2 : \texttt{Salary\_File}, PC : \texttt{Auditing}]$. The context describes the snapshot in the system when the user u_1 is requesting access to the resource `Salary_File` for auditing purposes. The organizational policy should be evaluated at this context before allowing or denying the request.

2.3 Context-awareness

Contexts that characterize external awareness are constructed automatically from the information gathered by the sensors and user stimulus to the system. When a stimulus is received by the system, the sensed dimensions might include LOC, $TIME$, WHO, WHY which correspond respectively to the location from where the system is accessed, the date/time at which the access happens, the user (role) of the subject accessing the system, and the reason for requesting the service. Context, as raw information gathered by sensors and other system stimuli, are aggregated into context representations by the context toolkit [13]. We call such contexts as *external contexts (EC)*. The twin issues related to the set EC are (1) the contexts are constructed in anticipation of future behavior, and

(2) the system reactivity is according the adaptation policies set by the system. As an example, consider external system attacks against which the system should adapt. For such anticipatory adaptation the system should construct *security* contexts from information available in its environment. This might include classes of people, their locations, sets of remote systems, and types of communication links. Based on this information the system constructs dimensions, and tag types for each dimension. Reactive adaptation is triggered when a specific situation arises.

The sets IC and EC should have a *bridge* in order to help construct internal contexts from external contexts. We assume that an ontology exists or the dimension names and their tag types are the same for constructing the IC and EC context types.

3 Adaptation

The system administration is driven by policies, which include rules relevant to accessing the system from its environment and rules for accessing its resources. Adaptation policies are rules. An adaptation rule r can be formalized as a pair (α, A), where α is the situation and A is an action. The meaning is that the rule r must be applied in a context c if $\mathbf{vc}(c, \alpha)$ is true, and the application of the rule requires the fulfillment of the action A. As explained earlier the situation α governing a rule can be evaluated at different internal as well as external contexts constructed dynamically during system evolution. Authorizing access, performing obligatory actions, and providing services are examples of actions to be performed in context c upon the validity of the situation α in context c. Thus, our semantics of adaptation satisfies both anticipatory and reactive adaptations. Given this semantics we suggest a few rules of inference for reasoning with adaptation policies.

3.1 Inference Rules for Adaptation

In an imperative programming style we can write an adaptation policy as `do` A `if` α `is true in context` c. In logic programming style we can write $A \Leftarrow \mathbf{vc}(c, \alpha)$. This later style is more conducive to express a set of inference rules for adaptation. Note that in our discussion the symbol $\rightarrow$ denotes 'logical implication', and the symbol $\Leftarrow$ denotes 'the imperative command `do`'.

Rule 1. If α is true in a context c then it is true in every context that includes c. A context c' that includes c is an *outer* context of c. This rule is necessary to ensure that an adaptation that is valid in a context remains valid in its enclosure. Without this rule, contradictions might arise in executing incremental adaptations. We write this formally as follows

$$\frac{A \Leftarrow \mathbf{vc}(c, \alpha)}{\exists c' \bullet A \Leftarrow \mathbf{vc}(c', \mathbf{vc}(c, \alpha))} \tag{1}$$

Rule 2. We use the Law of Excluded Middle to give two adaptation rules. There exists an adaptation rule that states that action A can be initiated in context c whenever α is true in context c. If it is true that in context c, $\alpha \rightarrow \beta$ is also true then we can apply action A in context c whenever β is true. This follows from the predicate logic modes ponen rule applied to context c.

$$\frac{(A \Leftarrow \mathbf{vc}(c, \alpha)), \mathbf{vc}(c, \alpha \rightarrow \beta)}{A \Leftarrow \mathbf{vc}(c, \beta)} \tag{2}$$

In the second case, there exists a rule that S can be initiated in context c whenever $\alpha \rightarrow \beta$

$$\frac{(A \Leftarrow \mathbf{vc}(c, \alpha \rightarrow \beta)), \mathbf{vc}(c, \alpha)}{A \Leftarrow \mathbf{vc}(c, \beta)} \tag{3}$$

Rule 3. Adaptations corresponding to conjunction and disjunction of situations are easy to construct. An example rule to deal with situation conjunctions is

$$\frac{(A_1 \Leftarrow \mathbf{vc}(c, \alpha)), (A_2 \Leftarrow \mathbf{vc}(c, \beta))}{(A_1 \circ A_2) \Leftarrow \mathbf{vc}(c, \alpha \wedge \beta)} \tag{4}$$

where $\circ$ means that both actions A_1 and A_2 should be performed, however they may be performed in any order. We give two adaptation rules enabling to infer adaptation actions across different contexts.

Rule 4. Let there be an adaptation rule for initiating action A in context c, provided α is true in context c. Assume that it is also true that whenever α is true in c, α' is true in another context c'. The rule below states that action A can be initiated in context c'.

$$\frac{(A \Leftarrow \mathbf{vc}(c, \alpha)), (\mathbf{vc}(c', \alpha') \ \rightarrow \ \mathbf{vc}(c, \alpha))}{A \Leftarrow \mathbf{vc}(c', \alpha')} \tag{5}$$

The above rule can be generalized to multiple adaptations.

$$\frac{(A_1 \Leftarrow \mathbf{vc}(c, \alpha)), (A_2 \Leftarrow \mathbf{vc}(c', \alpha')), \mathbf{vc}(c', \alpha') \ \rightarrow \ \mathbf{vc}(c, \alpha)}{(A_1 \circ A_2) \Leftarrow \mathbf{vc}(c', \alpha)} \tag{6}$$

The ability to automatically detect changes, construct contexts, and evaluate situations, combined with the ability to trigger policies at different contexts make our approach very powerful. The adaptation inference rules can be applied diligently to reduce the overload involved in choosing the right adaptations for different contexts.

4 Multi-agent System - Structure and Analysis

In this section we discuss an agent architecture and explain how they collectively cooperate in the system to improve awareness and facilitate adaptation.

4.1 Agent Architecture

We define agent types, instead of agents. An agent type consists of agents which have identical role and behavior. All agents are black-box agents. Many agents can be generated from each agent type. Agents belonging to two different types have distinct behavior. We use the notation $a : T$ to mean a is an agent of type T. The agent types and the relationships among agents of different types are enumerated below.

1. *CA Type:* Agents of this type will construct and manage contexts in a specific domain, according to the context calculus implemented in the context toolkit [13]. The type CA includes expert agents who can convert context types. The functionality of an expert agent is somewhat similar to the services provided by *Jena 2* ontology system [7].

2. *DA Type:* The system might have several database(s), each storing policies in a specific domain. Security policies, adaptation policies, and work flow policies belong to different domains. Type DA denotes the set of database agents, such that one agent is in charge of policies in one domain.
3. *MA Type* Autonomous agents who continuously monitor the internals of the system are of this type. An MA agent is a memory bank, tracking history of actions initiated by and performed by different agents in different contexts. They will feed valuable information for other agents to adapt.
4. *IA Type:* Each sensory device is an agent of type IA. Agents of this type are autonomous agents that focus on external information gathering. They aggregate raw data, and communicate with agents of type CA.
5. *RA Type:* All reactive agents are of this type. They receive stimulus from the environment, context information from CA agent types, and interact with the environment to provide the services that are allowed by the adaptation rules. When it receives a stimulus the context associated with the stimulus is sent to a context toolkit agent. It gets back the context constructed for that stimulus.
6. *TA Type:* Agents of type TA have the knowledge of adaptation rules and the ability to reason with them at different contexts. A TA type agent will get the adaptation policies from DA type agents, and dispatch the adaptation action to agents of type RA.
7. *PA Type:* Agents of this type guard the resources against unauthorized access and regulate the granting of resources to subjects in different contexts according to the satisfaction of system policies. Each agent may be in charge of guarding resources of any one type.
8. *BA Type:* Agents of this type guard the boundary of the system, such as firewall. Agents of this type act as bridges between the *outside* and *inside* of the system. They enforce the fire wall policies for all requests and services crossing the boundary.

The set EC of external contexts are constructed by agents of type CA at the request of the agent types $\{IA, RA\}$. The set IC of internal contexts are constructed by agents of type CA with the assistance from agent types $\{BA, MA, PA\}$. Agents of type BA, in collaboration with agents of type CA will be able to transit between EC and IC. An agent of type DA is a slave for the authorized agents of types $\{TA, PA, MA\}$ for providing access to the policies stored in it. Agents of type TA determine adaptation activity for a context by looking up with databases MA and by applying reasoning rules relevant to a current situation. Agents of type MA will monitor all activities that happen in the system, and feed the current system context to TA agents. Thus, collectively the agent types enumerated above will act in covering the adaptation requirements of the system.

4.2 Adaptation Protocol

Assume without loss of generality that there are many agents of each type in the system, distributed if necessary. We do not get into the specifics of agent communication, instead we discuss a high level interaction scenario in which the agents collaborate to adapt and react to specific requests from the system environment.
Step 1: Collect External Context Information- An agent $ia : IA$ corresponds to one sensory type of information. An agent $ra : RA$ interacts with clients external to the

system and collects information corresponding to the identity or role (WHO), the service ($WHAT$), the place where it is to be delivered ($WHERE$), at what time/date the service is to be delivered ($WHEN$) etc;. Until the ra is able to collect such information, the information automatically sensed for that session by ia will be repeatedly sensed. Both ia and ra forward the collected information to $ca : CA$. It is the responsibility of ra to react to the specific request S (stimulus) received by it.

Step 2: External Context Construction- Context information received may be fuzzy, especially for sensory data. Consequently, this step may require many iterations between ca and agent ia, until agent ca is able to determine the dimensions and the types associated for the context. Eventually, agent ca constructs a context c_e for stimulus, and a context d_e where response is expected, and sends to ra the tuple (d_e, c_e, S).

Step 3: Boundary Analysis- Agent ra forwards the pair (c_e, α) to agent $ba : BA$. The boundary agent ba forwards to agent $ca : CA$ the context c_e and the dimensions of interest to the system boundary. Interesting dimensions at the boundary might include role of client, client category, and purpose of use. Agent ca constructs from c_e a 'boundary context' c_b and an internal c_i and returns them to agent ba. This is possible because of our assumption that cas have semantic expertise and also have ontologies. We skip the actual details involved for such a construction. The boundary agent ba applies the fire wall security policy β for context c_b. That is, it evaluates $\mathbf{vc}(c_b, \beta)$. If the authentication fails, agent ba returns context c_e to agent ra. In turn ra will inform the authentication failure to the client from whom the stimulus was received. If the validation is successful, agent ba saves c_e and forwards service stimulus S with contexts d_e and c_i to $ma : MA$.

Step 4: Resource Requirements- The agent $ma : MA$ is autonomous and has a supervisory role in the system. In that capacity it can track data and communication traffic, monitor resource protection agents, and different agent interaction scenarios. In particular agent ma can get from $da : DA$ the history of requests serviced in every context. So, ma acts as follows:

- It sends the current execution environment as context situation and c_i to ca and receives c_i', where c_i' is the smallest enclosure [13] containing c_i.
- It requests da and receives from it the set of policies $R = \{r :: (A \Leftarrow \eta) \mid \mathbf{vc}(c_i', \eta)\}$ that are relevant for context c_i'. It computes the set R', $R' \subseteq R$, such that all actions specified in the rules of R' are necessary to fulfill the stimulus request S. This step is essentially a query processing steps, whose details are not important for us now.
- If $R' = \emptyset$, then the ma sends the message 'service cannot be fulfilled now' to agent ra, which in turn will inform its client outside the system.
- If $R' \neq \emptyset$ then for each action A in $r \in R'$ it determines the set of resources Q_A required by it. Then it determines the set of resource agents $Q_{pa} = \{pa_q \mid pa_q : PA, pa_q \texttt{ protects } q \in Q_A\}$.
- It requests the resource agents in the set Q_{pa} to evaluate the access rights for the role specified as part of context c_i'.
- If a resource is denied, the resource agent in charge of it communicates the decision *grant* otherwise communicates the decision *deny* to agent ma.

Step 5: Analysis by ma Once decisions are received from all pas, agent ma reasons about their decisions. If the service was successfully authenticated at the fire wall, but access to resources is denied at the context c_i' then ma can conclude that the role of subject or

CLASS (a dimension in BSC context) to which the subject was classified at the fire wall might have changed. Agent ma records this information, and informs agent ba of the denial of resource access for servicing the request S. If all the resources are granted, ma forwards contexts c_i and d_e and the set of rules relevant for servicing α to an adaptation agent $ta : TA$ requesting it to process the service adaptations.
Step 6: Informing Client of Service Denial- Agent ba forwards the contexts c_i and c_b to the reactive agent ra and informs the reason for service denial. In turn, the ra agent informs its client that the requested service cannot be provided because access to resources are denied.
Step 7: Adaptation- In order to adapt, agent ta has to select the adaptation rules from the set R'. There are three ways in which agent ta can resolve an adaptation.

1. *Specific Reactions* For every rule $r :: (A \Leftarrow \eta) \in R'$, it evaluates $\mathbf{vc}(d_e, \eta)$. If it is true then it selects A for adaptation. Once this is done for all rulers in R' there may be zero or more such adaptive actions. These are communicated to ra for execution.
2. *Reasoned Actions* When there is no direct match with an adaptation rules, an inference rule (from Section 3.1) must be invoked. The following are possibilities.
   - It is known that $\mathbf{vc}(d_e, \alpha)$ is true for some α and it can be proved that $\eta \rightarrow \alpha$ and there exists a rule $r \in R'$ such that $r :: (A \Leftarrow \eta)$. By Rule 2 in Section 3.1 adaptation action A can be initiated.
   - There exists rules $r \in R'$ such that $r :: (A \Leftarrow \eta)$, and $r' \in R'$ such that $r' :: (A' \Leftarrow \eta')$, and $\mathbf{vc}(c_e, \alpha)$ is true where $\eta \wedge \eta' = \alpha$. By Rule 4 in Section 3.1 adaptation actions A and A' can both be initiated.
   - It is known that $\mathbf{vc}(d_e, \alpha)$ is true for some α, the rule $(A \Leftarrow \mathbf{vc}(c', \eta))$ exists in R, and it can proved that $\mathbf{vc}(c_e, \alpha) \rightarrow \mathbf{vc}(c'_i, \eta)$ then initiate adaptation A. This is justified by Rule 6 in Section 3.1.

An adaptation need not be successful. When the effort to adapt fails, agent ta informs ma of the failure, and the reason for the failure. In turn ma informs ra through ba. In case an adaptation is successful, both ra and ba are given the set of adaptation actions.

5 Conclusion

Self-adaptation is a must for autonomic computing systems. Most of the current work have only limited capabilities. An example is the Personal View Agent (PVA) whose design is based upon some empirical calculations tracking user profiles. As brought out in [5] there is a great challenge ahead in developing self-adaptive systems based on sound software engineering principles. To the best of our knowledge the contributions in this paper are new. In particular, the proposed logic of contextual adaptation has the potential to immensely increase the adaptation capabilities of any context-aware software system. By importing it to a multi-agent system architecture we only increase its enforcement power, because we can add the intelligence inherent for agents with the ability to reason using the adaptation logic. Within the adaptation logic the traditional $\langle M - A - P - E \rangle$ activities can be embedded as rules. Thus, the adaptation logic is both an abstraction and a generalization of the traditional adaptation activities. The additional merits of our approach are the following:

1. By designating roles for agents, we have separated design from implementation issues. Context implementation, for example, is independent from how it will be used. Defining situations formally we are able to evaluate situations in different contexts, which in turn make adaptations truly dynamic.
2. Adaptation logic is simple for the present. Experts may be able to add more adaptation rules. Actual domain specific rules, such as security and business work flow policies, may be added/deleted independent from the rules of inference for adaptation.
3. As opposed to the looping activities of $\langle M-A-P-E\rangle$ system [12], many adaptation strategies such as parallel or sequential, allowed by the policies and the logic are possible.
Some of the challenging issues for further research include the application of the proposed architecture to a large case study, developing an implementation framework for orchestrating different sequences of multiple adaptations, and assessing the effectiveness of the proposed architecture.

6 Acknowledgments

This research is supported by Research Grants from National Natural Science Foundation of China (Project Number 61103029), Natural Science Foundation of Jiangsu Province, China (Project Number BK2011351), Research Development Funding of Xi'an Jiaotong-Liverpool University (Project Number 2010/13), and Natural Sciences and Engineering Research Council, Canada.

References

1. S. A. Hnaide. A Framework for developing Context-aware Systems. Master of Computer Science Thesis, Concordia University, Montreal, Canada, April 2011.
2. IBM White Paper. *Autonomic Computing Concepts*, Available at http://www-03.ibm.com/autonomic/pdfs/AC_Concepts.pdf
3. D. F. Bantz et al. *Autonomic personal computing.* IBM Systems Journal, Vol. 42, No 1, pp. 165-176, 2003.
4. C.C. Chen et al. *PVA: A Self-Adaptive Personal View Agent System.* ACM SIGKDD 2001, San Francisco, CA, USA.
5. B. Cheng et al. *Software engineering for self-adaptive systems: A research roadmap.* in Software Engineering for Self-Adaptive Systems, LNCS vol. 5525, 2009.
6. Paul Horn. *Automatic Computing: IBM's Perspective on the State if Information technology*, IBM Corporation, October 15, 2001.
7. Hewlett Packard. *HP's Darwin Reference Architecture Helps Create tighter Linkage Between Business and IT*, San Jose California, May 6, 2003, available at: http://www.hp.com/hpinfo/newsroom/press/2003/030506b.html
8. N. Ibrahim, V. Alagar, and M. Mohammad. *Managing and Delivering Trustworthy Context-dependent Services*. The 7th International Workshop on Service-oriented Applications, Integration, and Collaboration (part of the 8th IEEE International Conference on e-business Engineering), October 19-21, Beijing, PRC.
9. Microsoft Corporation. *Microsoft Dynamic Systems Initiative*, White Paper, October 2003, available at: http://download.microsoft.com/download
10. Richard Murch, *Autonomic Computing,* Prentice Hall Professional Technical Reference, IBM Press, pp. 235-245, 2004.

11. Sun Microsystems, *ARCO, N1 Grid Engine 6 Accounting and Reporting Console*, White Paper, May, 2005, available at: http://www.sun.com/software/gridware/ARCO_whitepaper.pdf
12. P. Vromont, D. Weyns, S. Malek, and J. Anderson. *On Interacting Control Loops in Self-Adaptive Systems.* In Proceedings of SEAMS'11, Waikiki, Honolulu, HI, USA, May 2011.
13. K. Wan. A Brief History of Context. *International Journal of Computer Science Issues*, Volume 6, Issue 2, Nov 2009.