*Wireless Sensor Network*, **2009, 1, 1-60**
Published Online April 2009 in SciRes (http://www.SciRP.org/journal/wsn/).

# Policy Based Self-Adaptive Scheme in Pervasive Computing

**Jian Quan OUYANG#*1, Dian Xi SHI#2, Bo DING#3, Jin FENG※4, Huai Min WANG#5**
*#1School of Computer, National University of Defence Technology, Changsha, China*
**College of Information Engineering, Xiangtan University, Xiangtan, China*
*※Nanchang Military Academy Unit Battle, Nanchang, China*
*E-mail: 1kissingman1@gmail.com, 2dxshi@nudt.edu.cn, {3maildb, 5whm_w}@163.com, 4fengjin0510@126.com*
*Received February* 15, 2009; *revised March* 6, 2009; *accepted March* 10, 2009

## Abstract

Nowadays, application systems in pervasive computing have to be self-adaptive, which means adapting themselves to dynamic environments. Our aim is to enable systematic development of self-adaptive component-based applications. The paper first introduces a novel policy based framework for self-adaptive scheme in pervasive computing. Then the proposed policy ontology and policy language are well expressive and easily extensible to support the design of policy which is based on the Separation of Concerns principle. Furthermore, the context-driven event channel decouples the communication between the suppliers and consumers for asynchronous communication. The proposed framework can provide both a domain-independent and a flexible self-adaptation solution.

**Keywords:** Policy Ontology, Self-Adaptive, Policy Language, Pervasive Computing

## 1. Introduction

Technology of software evolution drives the need for software self-adaptive. Moreover, while pervasive computing environment is open and dynamic, application systems in pervasive computing have to be self-adaptive, which is adapt themselves to work in dynamic environments. Previous adaptation work is based on predicting future circumstances and adapting themselves by way of embedding the adaptation decisions in the program code. It is clearly that it is done in an ad hoc way. While policy can define the behaviour of adaptive are applied by different research projects for the flexible reconfiguration systems, it seems that a feasible approach to be decoupled from functional concerns and systematically develop self-adaptive applications. Moreover, as it can separate the business logic (rules) from the controls (programming code) of the implementations, policy-based scheme are typically more flexible and adaptable than non- policy-based approach. In a word, policies can specify and adapt the behavior of a system and can be applied to various areas: auction mechanisms, access control, Privacy (Information Collection Policies), Context aware computing, etc.

In this paper, we present a policy based adaptive architecture for pervasive computing. Different from current policy approach, in the view of the proposed scheme, the context information is used as meta data and the policy is applied to meta protocol, thus it can materialize a reflective approach in the adaptation architecture. In additional, the proposed policy ontology and policy language can support for knowledge representation and reasoning and knowledge sharing. And they are feasible to support the design of policy which is based on the Separation of Concerns principle.

The rest of the paper is structured as follows. In next section, we introduce the current state of the art. Section 3 discusses the requirements of self-adaption. Section 4 describes the overview of adaptation architecture. Following this, Section 5 proposes a policy descriptive language for pervasive computing. Section 6 illustrates the event scheme. Section 7 will give an introduction to the prototyping applications in fire alarm scenario and preliminary experiments. Finally we summarize our work and give future plan in Section 8.

## 2. Current State of the Art

There are several ways for proposing polices. Previously, the approaches to policy specification are proposals for policy language specification. Lobo [1] depicted the PDL (policy description language) to describe the strategies for mapping a series of events into a set of actions. Damianou [2] described a policy language (Ponder) applying for both management and security policies for distributed systems. Anthony [3] introduced a policy definition language which is designed to permit powerful expression of self-managing behaviours. Moreover, a prototype library implementation of the policy support mechanisms which can facilitate adaptive-policy deployment is illustrated. Ahn [4] proposed a high-level policy description language for formally specifying context entity relation, and introduced the translator which can provide automatic generation of Java classes for ubiquitous entities.

The other approach is based on logic programming for supporting well defined semantic. Semantic Web Languages for policy specification: KaoS [5] and Rei [6]. Uszok [5] proposed a framework for specification, management, conflict resolution and enforcement of policies which is used OWL ontology. Kagal [6] introduced a policy language (Rei) for pervasive computing environment which can express the behaviour of entities and it is used as part of a secure pervasive system.

Recently, there are several policy based applications in the ubiquitous/pervasive computing scenarios. Rukzio [7] presented policy based adaptive services for mobile commerce, but the event scheme is not mentioned. Erradi [8] introduced policy-based middleware, Manageable and Adaptive Service Compositions (MASC), for dynamic self-adaptation of Web services compositions. David [9] presented an adaptive framework which is based on the Fractal component model. In the framework context-awareness service can provide information about the execution context. Chan [10] proposed an event model for a highly adaptive mobile middleware, Web Proxy for Active Deployable Service (WebPADS). Bandara [11] applied Event Calculus to transform both policy and system behaviour specifications into a formal notation. However, these methods did not concentrate the Separation of Concerns principle to support reconfiguring system based on reflective scheme.

Lately, Adamczyk [12] proposed a lightweight framework called the Autonomic Management Toolkit, which can support dynamic deployment and management of adaptation loops.

## 3. Requirements of Adaptation in Pervasive Computing Environment

Generally, self-adaptive applications need to control how and when decisions and actions are taken. Policy-based scheme can specific the adaptation layer and adaptation time [13]. Different from the three basic requirements (Uniformity, Separation and Generic) for the development of adaptation architecture [7], we define the following three basic requirements of policy in pervasive computing environment:

-**Expressiveness:** The first requirement is that suitable expression of policies is important for describing the rules to specify the behavior of a system. On the one hand, it is need to be restricted to avoid ambiguities or ill-defined policies. On the other hand, it can not be too complex for untrained user to write rules.

-**Well-defined semantics:** The next requirement is well-defined semantics. Obviously, Well-defined policy can support for knowledge representation and reasoning and knowledge sharing of polices. Moreover, it can enable interoperability of heterogeneous rules.

-**Usability:** As the perspective pervasive computing is to seamless integration of computing into the user's everyday life, make it easy for users to write rules is one of the critical requirement of policies. Make rules intelligible to the common user and declarative, human readable interface is favourable for design polices

-**Lightweight:** Foe the reason of limitation of resource in pervasive computing environment, strong rule engine is difficult to run for the various devices in pervasive computing environment. Lightweight policy architecture is necessary for devising the rule engine.

## 4. Overview of Adaptation Architecture

### 4.1. Core Idea

The core idea of the adaptation architecture is shown in Figure 1. The architecture is based on the tenets of policy-driven systems which are applied in various adaptive systems.

We are using policies which can be seen as a set of sophisticated rules modelled by Event-Condition-Action rules for the definition of the adaptive behaviour in pervasive computing environment. Thus it can react to changes of the context information by reconfiguring the application.

Our proposed policy engine is based on the Separation of Concerns [14] principle: extract explicit rules of business logic from various applications. In the first step in a cycle, Context data is provided by context-aware component, PolicyController matches all polices with the Context data and select the appropriate policy. Then judge whether the conditions in the "action-event" table are met. If they are met, EventMonitor triggered by relevant events and notify PolicyExecutor. As the next step PolicyExecutor will executes the predefined rules and lead to a change of the context information.

### 4.2. Reflective Scheme

A reflective scheme can ensure that can support structural reconfiguration while examining and change environment, aiming to self-adapt at runtime. As shown in Figure 2, in our architecture, the strategy of the separation of component and policy can gain the decoupling of meta-level scheme and based-level implementation. The advantage of the reflective scheme can conclude two parts:

1) The policy-based application system can be flexible, extensible and adaptive, since the policy can be deployed and modified in the course of runtime of systems.

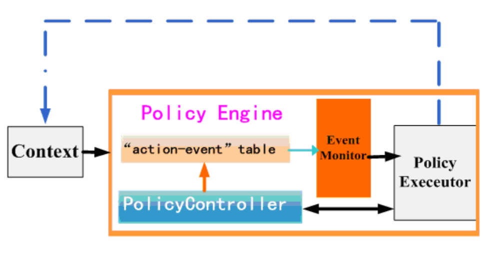

**Figure 1. The core idea of the adaptation architecture.**

2) Policy-based scheme decouples the reusable component between the developments and deployability stage. The developer can only focus on using policy to describe the base-level business logic, and the deploymentor can designate the component according to the application environment.

Context is the provider of the meta-level data, policy is the meta-level protocol between the business logic and context, policy engine is meta-level procedure, and behaviour component is the base-level computing entity.

### 4.3. Context-driven Mechanism

Context is one of the most important features of pervasive computing. As the dynamic character of pervasive computing, it is necessary to model and specify context in a way such that context information can easily exchange, share and reuse their knowledge. For simplicity, we define context as four tuple ConData=(ConSup, ConType, Value, TimeStamp), ConSup is the supplier of context data, ConType indicates the type of the context (e.g., location, temperature), Value gives the content of the context data, Timestamp describes the generation time of the context.

Here, we classify the components into two classes: Context-aware components which gain and aggregate the context data and Behaviour components which carry out the actions according to the predefined rules in policy engine. The policy engine is driven by policies which are a set of rules in XML files and describe how the behaviour component reacts in a specific context to support deployable application. As shown in Figure 2, context is the only starting point of self-adaptation and also the end point of adaptation.

### 4.4. Context-driven Policy Based Framework

The model of context-driven policy consists of three layers, which is shown as Figure 3. The bottom layer is context layer. The top layer is self-adaptive layer, while the policy layer is in the middle. Context layer can abstract the state of physic and information space in pervasive computing environments and context-driven events. The policy layer is used for describing self-adaptive rules including context constraint, description of actions. Self-adaptive layer is based on context-driven scheme. Context based event is the jumping-off point of the course of adapt procedure and the sole driver for adapt procedure.

## 5. Policy Descriptive Language for Pervasive Computing (PDLPC)

### 5.1. Policy Ontology

To attain better semantic language understanding and share knowledge for reusable, Figure 4 illustrates the policy ontology we are developing to express the structure of polices precisely.

The proposed policy ontology defines the vocabularies for indicating rules that perform different types of actions. To describe policy rules, the ontology define the basic concepts of "policy ontology" including "Priority", "Event", "Precondition" and "LogicType". Furthermore, we use "Unionof" relation to design the hierarchical structure of the policy ontology.

The structure of the policy ontology is as follows.

**Priority:** The "Priority" class defines the priority between policies. It has been further classified into "High" and "Low" subclass.

**LogicType:** LogicType class indicates types of logic including two-valued logic and fuzzy logic.

**Precondition:** Preconditions are constraints on the action and environment. We use "Unionof" relation to model the composition of the value restriction.

**Event:** The "Event" class implies the policy is triggered by the changed environment context. The "Event" class include:

1) "EventTpye" subclasses comprise "AtomEvent" and "Composite" subclasses. In the meantime, "Composite" subclass is composed by "EventOperator" and "AtomEvent" via "Unionof" relation.

2) "LogicTpye".

3) "Precondition".

4) "Component" subclasses can indicate the related component which can perform a specific action.

5) "Action" subclasses can represent an invocation to certain type of computing procedures to acquire user information or provide services in the pervasive environment.

Also, the "Unionof" relation can describe the "EventTpye", "LogicTpye", "Precondition", "Component" and "Action" to form "Event" class.

### 5.2. Policy Descriptive Language for Pervasive Computing (PDLPC)

Policies can be described at different levels of abstraction. At a high level, Policies could be specified using natural language. At a low level, the method of logic or algebraic can be applied to specify policy description. In the intermediate point, production rules are be found to specify policies. In this paper, we prefer in the intermediate point for effectively computing in pervasive computing

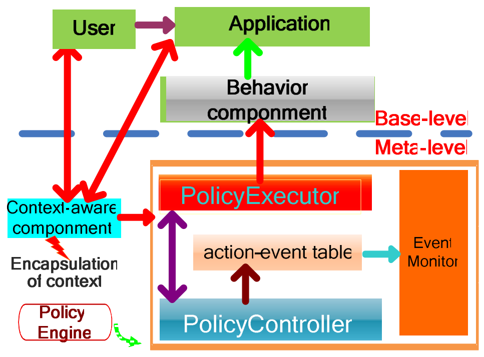

**Figure 2. Reflective scheme of context-driven policy scheme.**

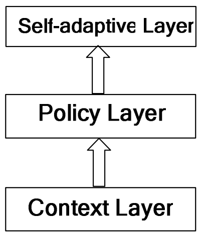

**Figure 3. Model of context-driven policy.**

environments. For this reason we propose a Policy Descriptive Language for Pervasive Computing (PDLPC) based on the proposed policy ontology.

The syntax of PDLPC is defined based on the BNF notation. The most important features of BNF used in this paper are as follows:

• =is the defining symbol. On the left-hand side is the name of the grammar rule and on the right-hand side is the definition of that name.

• | indicate optional elements.

• {and} indicate repetition. Zero or more elements.

• , is the definition separator symbol. It separates alternatives in a grammar rule.

• ; is the terminator symbol. Every rule is terminated by this symbol.

The definition of PDLPC is as follows.

<PolicySet>::={<Policy>};
<Policy>::= <PolicyID>,<Priority>,<LogicType>, <EventPreconditionGroup>,<Event>;
<Priority>::=<High>|<Low>;
< LogicType >::=<Two-Valued >|<Fuzzy>;
<EventPreconditionGroup>::= {<EventPrecondition>};
<EventPrecondition>::= <EventPreconditionid>, <EventCondition>,<Restriction>;
<EventCondition>::=<Context>|<State>;
<Context>:==<ContextTime>,<ContextAattribute>,<DataType>;
<DataType>:==<int>|<char>|<float>|<double>|<datetime>;
<Restriction>::=<LogicOperator>,<Value>;
<LogicOperator>::=<Over>|<Below>|<Equate>;
<Event>::=<EventType>,<LogicType>, <PreconditionGroup>,<Component>,<ActionGroup>;
<EventType>::=<AtomEvent>|<CompositeEvent>;
<AtomEvent>::=<ContextValueEvent >|<StateEvent>;
<CompositeEvent>::=<AtomEvent>,<EventOperator>;
<EventOperator>::=<→>|<and>|<or>|<not>|;
<LogicType>:=<Two-valued>|<Fuzzy>;
<PreconditionGroup>::={<Precondition>};
<Precondition>::=<PreconditionID><Context>,<Restriction>;
<ActionGroup>::={<Action>};
<Action>:: =<ActionID>, <Component>, <Method>, <ParameterSet>;

PDLPC consists of there layers: policy-event-action. Policy is at a high level and could be a set of rules which govern the behaviour of a system can be triggered by events. At a low level, Action is a domain dependent action and Precondition is constraints on the Action and Component. It can reveal the execution of actions and consider greater understanding of the action and its parameters. In the middle level, Event is used to trigger policy and represent the execution of action reacts in a specific context.

From the above description, it is convenient to be able to define polices separately, and re-use them via PDLPC.

## 5.3. XML Based Representation

For rules are intuitive and natural way of thinking, there is need to write rules conveniently. As XML becomes the de facto standards for data representation and interchange, and XML data which can be viewed as a hierarchically-structured rooted tree is convenient for represent the policy descriptive language. Here, we prefer to use XML-based representation for PDLPC.

We use the fire alarm example to illustrate PDLPC using XML, as show in Figure 5. From the example, we can find the useful features of our approach.

1) Policy is a hierarchically-structured that can be favourable to XML parser.

2) Events and parameters are attached to a component.

## 5.4. The Lifecycle of Polices

The lifecycle of polices is as shown in Figure 6. It includes the main steps and related activities in the policy life cycle.

The step of policy analysis is to parse the policy set via XMLParser. In the meantime, for the efficiency and simplify of policy management, priorities of policies fall into two main categories: low and high. The conflict between the two policies can be resolved at run-time.

When PolicyController check the policy is high priority, the policy will be activated, otherwise it will be deactivated. According to the reflective scheme, application developer can adjust relevant policies to the new situation including insert, modify and delete policy in Policy Set. The policy maintenance mechanism is convenient to improve the policy definition and deployment.

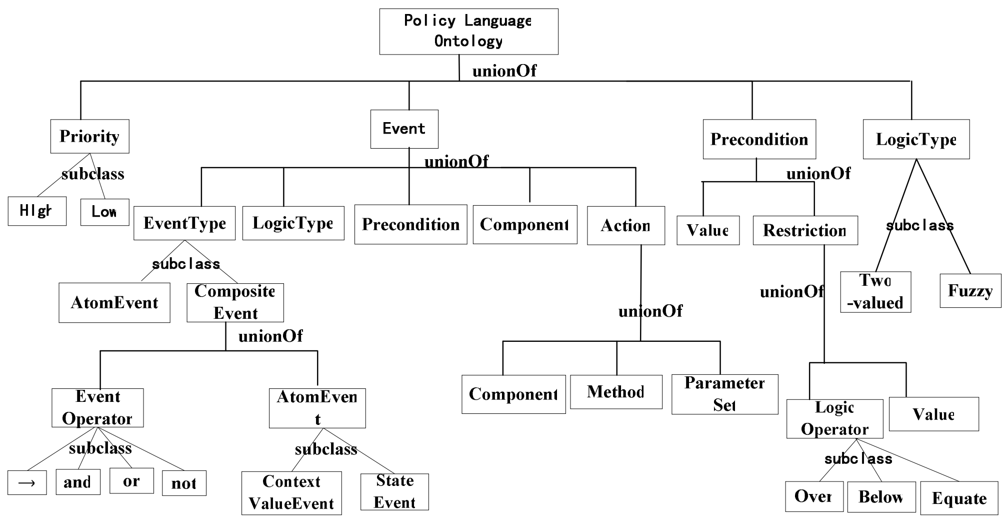

**Figure 4. Policy ontology.**

```
      <PolicySet PolicyID="FireAlarm">
         <Policy>
            <Priorities>High</Priorities>
            <LogicType>Two-Valued</LogicType>
            <PreconditionGroup>
               <Precondition PreconditionID="First">
                  <Context>
                     <ContextTime>180506</ContextTime>
            <ContextAattribute>temperature</ContextAattribute>
                     <DataType>float</DataType>
                  </Context>
                 <Restriction>
                     <Over>
                        <Value>200</Value>
                     </Over>
               </Restriction>
            <Component>Temperature_sensor_demo</Component>
                 </Precondition>
               </ PreconditionGroup>
      <Event>
               <EventType>ContextEvent</type>
               < LogicType >IF-THEN</Type>
               < PreconditionGroup>
                 < Precondition PreconditionID="first">
                  <Context>
                     <ContextTime>180507</ContextTime>
                     <ContextAattribute>fog</ContextAattribute>
            <DataType>float</DataType>
           </Context>
           <Restriction>
              <Over>
                <Value>0.3</Value>
              </Over>
           </Restriction>
           <Component>Fog_sensor_demo</Component>
         </Precondition>
       </ PreconditionGroup>
       <ActionGroup>
           <Action ActionID="Fire_alarm">
             <Component>Fire_alarm_demo</Component>
             <Method>Forecast</Method>
               <ParameterSet></ParameterSet>
             </Action>
           </ActionGroup>
       </Event>
     </Policy>
  </PolicySet>
```

**Figure 5. Fire alarm example in XML.**

## 6. Event Scheme

### 6.1. Context-Driven Event Channel

The context-driven event channel decouples the communication between the suppliers and consumers for asynchronous communication. Event scheme supports asynchronous communication and lets one or more suppliers to send events to more than one consumers occurring at the same time. Context-driven event channel is as shown in Figure 7.

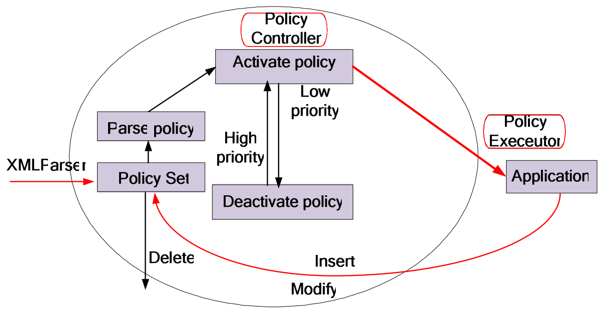

**Figure 6. The lifecycle of polices.**

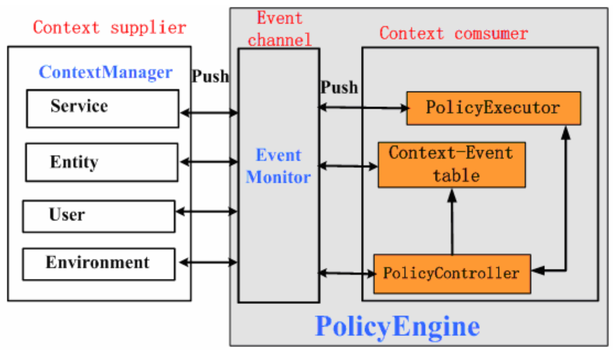

**Figure 7. Context-driven event channel.**

**Context Suppliers and Consumers:** ContextManager is the context suppliers consist of context data which is organized as hierarchical ontology including Service, Entity, User and Environment. When the context data is changed, context suppliers will push events to consumers. Context consumers which are managed by PolicyController are final goals of the events generated by the context suppliers pushing the events.

**Event Channel:** The event channel plays the role of a central mediator between the context consumers and suppliers. Both the suppliers and consumers connect to one or more event channels which are managed by EventMontior. An event channel is responsible for transferring events from the suppliers to the consumers.

**Reflective Scheme:** PolicyExecutor can dynamically change context data via EventMonitor for reflection. Moreover, when the policy in Policy Set is modified or deleted, it will lead to relevant change in Context-eventTable and EventMonitor.

### 6.2. Event Composition

An adaptation policy consists in a set of rules, each of the form Event-Condition-Action (ECA). The event can be classified into two classes: AtomEvent and CompositeEvent. AtomEvent consists of two types: ContextValueEvent represents the change of the context data, while StateEvent gives a clue to the state of the system.

There is four event operators that allow various kinds of complex events to be specified:→, and, or, not.

• →: If A and B are events, A→B denotes that event B must only be triggered after event A or that event A and B must be triggered in sequence.

• and: If A and B are events, A and B denotes that CompositeEvent is triggered when both event A and B have been happened no matter the occurrence time of event A, B.

• or: If A and B are event s, A or B denotes that CompositeEvent is triggered when either event A or B is happened.

• not: If A is a event, not A denotes that CompositeEvent is triggered when event A is not happened.

The event composition can be composed via simple Boolean expressions. The hierarchical event composition consists of multiple levels of atom events. The example

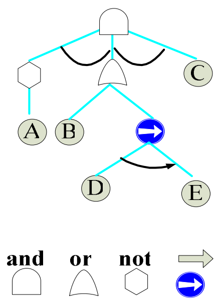

**Figure 8. Hierarchical event composition example.**

is as shown in Figure 8, The composition event F is composed by five atom events A,B,C,D,E: (not A) and (B or (D→E)) and C.

## 7. The Implementation of Adaptation Architecture

For the convenience of implementing the self adaption, we extend the CCM component container for supporting policy scheme. As shown in Figure 9, We augment the infrastructure of CCM component container including increasing the context list, Context-Event table, Policy-Controller, Policy Table and Policy Executor for supporting the parse and handle of policy. Context list is a two-dimension table, which consists of component name, context name and the value of the context data. Context-event table can describe the change of physical and information space. It includes the field of context name, event ID and event name. PolicyController is responsible for matching all polices with the Context data and select the appropriate policy. Policies are defined as a set of sophisticated rules which is described in XML (EXtensible Markup Language). The Policy Table can maintain the policy information which contains policy ID, policy priority, event ID, The reference of PolicyExecutor pointer. They indicate the execution of action reacts in a specific context and are storaged by hash table.

The functionality of Policy Engine is monitoring the change of the value of Context and executing the predefined polices. It comprises the Event Monitor, Policy Executor, PolicyController, Policy Table, Policy Parser and POA (Portable Object Adapter).

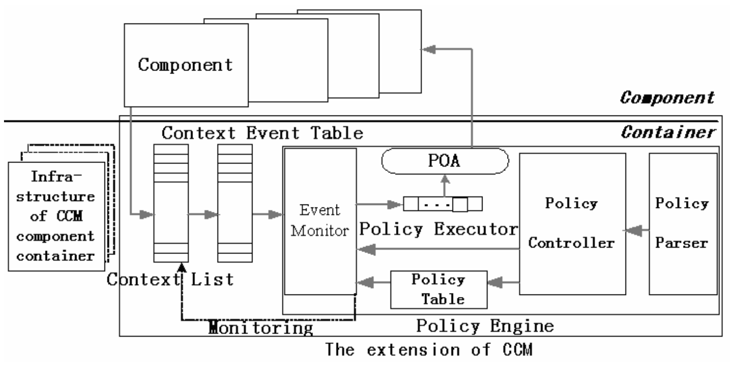

**Figure 9. Extension of CCM component container for component fault detection.**

◆ Policy Parser is a CORBA object, which is a XML parser. The functionality of Policy Parser is to match all polices in the Context Event Table and select the appropriate policy to dynamically generate the Policy Executor.

◆ Policy Executor is a two dimension pointer array. The first dimension is context name, and the second is a pointer which point to a group of CORBA objects and corresponding interfaces. The policy is storied as a structure of policy condition, action type, component name, method of component. Policy Executor can check the condition of policy is met, if it is true, the method of the component can be triggered by POA.

◆ PolicyController takes charge loading, uninstalling, activating/deactivating the polices. In the meantime, it can adjust the priority of the polices according the demand of the applications.

◆ Policy Table is initialized by PolicyController and can providing the query operation. For instance, it can retrieve the reference of the corresponding Policy Executor by event ID.

◆ Event Monitor is also initialized by PolicyController. The role of Event Monitor is to register the event to the Context Event Table or remove the event from the Context Event Table. The event ID is unique and can be bound to the Policy Table. It can periodically monitor the change of value the Context List. When the change is detected, it judge whether the context event is in the Context Event Table, if it is false, insert the context event into Context Event List.

## 8. Prototype Implementation

### 8.1. Fire Alarm Scenario

The first prototype implement is based on fire alarm scenario, as shown in Figure 9. There are temperature infrared sensor, sprinkler control valve and fire warning light in a room. The fire alarm application consists of sprinkler control valve component, fire warning light component, temperature-aware component and fire alarm policy. The temperature-aware component aggregate the context information from the temperature sensor and the aggregated data which is the occurrence likelihood of fire alarm (0%-100%) is displayed by fire monitor terminal. When the captured values from temperature sensor exceed the threshold, fire alarm component will drive the fire alarm lamp give off flashes of light and sprinkler control valve begin to sprinkle water which can be activated by sprinkler control valve component. As the predefined value which is assigned by policy stored in XML file is update from 200 to 150, it is need to only restart the fire alarm application without recoding the program code. Moreover, while the temperature exceeds the threshold, policy engine will result in a change of the context information by way of executing the predefined polices.

The XML parser is based on TinyXML parser (from SourceGauge Website). In the mean time, the register/recall mechanism is used in the communication between the temperature-aware component and context manager component.

### 8.2. Error Tolerant Policy

As pervasive computing environments is open and dynamic, the technology is sustainable and high confident if it is inconspicuous to the user and does not disturb the user's attention. This necessitates the pervasive computing system has to be resilient to faults and should be able to be error-tolerant.

Here, a prototype implementation of component error recovery has been realized based on fire alarm scenario.

The instance of error tolerant policies is as shown in Figure 11. It means that "If temperature context name=[CompomentStatus], and context value=[Component Failure], then call the activate () method of temperature-aware component".

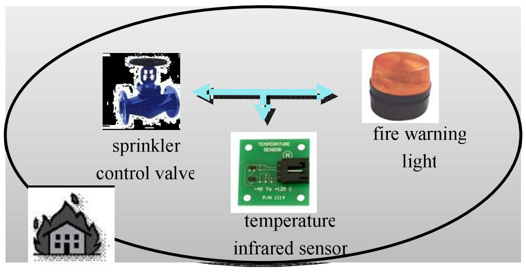

**Figure 10. Fire alarm scenario.**

```
  <PolicySet>
     <Policy>
        <Policy Description>Temerature error tolerent policy</Policy
Description >
        <Priorities>High</Priorities>
        <Event>
          <Event Description >Platform cotext event</Event Descrip-
tion >
          <EventType>ContextEvent</type>
          <Type>IF-THEN</Type>
          <PreconditionGroup>
             <Precondition>//Error event triggered
                <Contextid>
                  <Component>Temperature_sensor</Component>
                  <ContextTime>180506</ContextTime>//
                  <Attribute>ConponentState</Attribute>
                  <LogicType>Bool</LogicType>//
                  </Contextid>
                  <Restriction>
                     <Equate>
                       <Value>Failure</Value>//The Component is failed
                     </Equate>
                   </Restriction>
                  </Precondition>
           </PreconditionGroup>
           <ActionGroup>//action
                  <Action>
                    <Component>Temperature_sensor</Component>
                    <Method>activate</Method>//reload the component
                    <ParameterSet></ParameterSet>
                  </Action>
           </ActionGroup>
        </Event>
     </Policy>
  </PolicySet>
```

**Figure 11. Instance of error tollrant policy.**

### 8.3. Comparison with Current Methods

The proposed self-adapt model is based on reflective scheme for adaptive middleware support. It means that context information is used as meta data and policy can be regarded as meta protocol. Thus it can separate self-adapt functionality from business logic of a system. Compared with David [9], the proposed framework can be more reusable and flexible. Compared with Adamczyk [12], the proposed policy ontology is well defined semantic. This means that it can adapt the behaviour of applications in the pervasive computing without re-coding functionality, and a change in the applications can be applied without restarting the system. Moreover, the proposed policy language is based on the policy ontology, which has a common semantic understanding of aadaptive rules for well-defined semantics, thus it is well expressive and easily extensible to support the design of policy engine which is based on the Separation of Concerns principle.

However, there is still a limitation of the proposed scheme that has an impact on the complexity of component management because there both exist behaviour component in the base-level and context-aware component in the meta-level.

## 9. Conclusions

In this paper, we have presented policy based adaptive architecture for pervasive computing. The proposed policy ontology can support for knowledge representation and reasoning and knowledge sharing and integration for defining adaptive rules. Also, the proposed policy descriptive language for pervasive computing can enable define polices separately, and re-use them. Moreover, policy management allows application developers to ensure flexibility and adaptability. Furthermore, the policy mechanism is based on not only event- condition-action rules, but also more abstract utility/goal policies.

Now our ongoing work is to apply the adaptive architecture to the museum monitor scenario in China in practice.

## 10. Acknowledgment

This work was supported by the National High-Tech Research and Development Plan of China under Grant No 2006AA01Z198 and China Postdoctoral Science Foundation Grant No 20070420187.

## 11. References

[1] J. Lobo, R. Bhatia, and S. Naqvi, “A policy description language,” 16th National Conference on Artificial Intelligence, Orlando, Florida, USA, 1999.

[2] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, “Ponder: A language for specifying Security and management policies for distributed systems, V 2.3,” Imperial College Research Report DoC 2000/1, October 2000.

[3] R. J. Anthony, “A policy-definition language and prototype implementation library for policy-based autonomic systems,” 2006 IEEE International Conference on Autonomic Computing, pp. 265–276, 2006.

[4] J. Ahn, B. M. Chang, and K. G. Doh, “A policy description language for context-based access control and adaptation in ubiquitous environment,” Emerging Directions in Embedded and Ubiquitous Computing, pp. 650–659, 2006.

[5] Uszok, Bradshaw, Jeffers, Suri, Hayes, Breedy, Bunch, Johnson, L. Kulkarni, “KAoS policy and domain services: Toward a description-logic approach to policy representation, deconfliction, and enforcement,” in POLICY, pp. 93–96, 2003.

[6] K. Lalana, F. Tim, and J. Anupam, “A policy language for a pervasive computing environment,” Proceedings of the 4th International Workshop on Policies for Distributed Systems and Networks, Washington, DC, USA: IEEE Computer Society, pp. 63–74, 2003.

[7] E. Rukzio, S. Siorpaes, O. Falke, and H. Hussmann, “Policy based adaptive services for mobile commerce,” 2nd Workshop on Mobile Commerce and Services (WMCS 2005), Munich, Germany, pp. 183–192, July 19, 2005.

[8] P. C. David and T. Ledoux, “Towards a framework for self-adaptive component-based applications,” Proceedings of Distributed Applications and Interoperable Systems 2003, Lecture Notes in Computer Science 2893, pp. 1–14, November 2003.

[9] A. Erradi1, P. Maheshwari, and V. Tosic, “Policy-driven middleware for self-adaptation of web services compositions,” Middleware 2006, LNCS 4290, pp. 62–80, 2006.

[10] A. T. S. Chan, S. N. Chuang, J. N. Cao, and H. V. Leong, “An event-driven middleware for mobile context awareness,” The Computer Journal, 47(3), pp. 278–288, 2004.

[11] A. K. Bandara, E. Lupu, and A. Russo, “Using event calculus to formalise policy specification and analysis,” 4th IEEE International Workshop on Policies for Distributed Systems and Networks, pp. 26–39, 2003.

[12] J. Adamczyk, R. Chojnacki, M. Jarząb, and K. Zieliński, “Rule engine based lightweight framework for adaptive and autonomic computing,” International Conference on Computational Science 2008, LNCS 5101, pp. 355–364, June 23–25, 2008.

[13] P. McKinley, S. Sadjadi, E. Kaste, and B. Cheng, “Composing composing adaptive software,” IEEE Computer, 37(7), pp. 56–64, July 2004.

[14] W. Hürsch and C. V. Lopes, “Separation of concerns,” Technical Report NU-CCS-95-03, Northeastern University, Boston, Massachusetts, 1995.