Towards Self-managed Pervasive Middleware using OWL/SWRL ontologies

Weishan Zhang and Klaus Marius Hansen

Department of Computer Science, University of Aarhus
Aabogade 34, 8200 Århus N, Denmark
{zhangws, klaus.m.hansen}@daimi.au.dk

Abstract. Self-management for pervasive middleware is important to realize the Ambient Intelligence vision. In this paper, we present an OWL/SWRL context ontologies based self-management approach for pervasive middleware where OWL ontology is used as means for context modeling. The context ontologies are incorporating the dynamic context information, including device and service run time information, which can then be used for running status checking and diagnosis, QoS monitoring, and further to achieve other self-management features, such as the self-configuration and self-adaptation. We demonstrate the OWL/SWRL context ontologies based self-management approach with the self-diagnosis in Hydra middleware, using device state machine and other dynamic context information, for example web service calls. The evaluations in terms of extensibility, performance and scalability show that this approach is effective in pervasive service environment.

1 Introduction and Motivation

Context awareness[1] is one of the key features for pervasive middleware. It can provide the potential to improve the flexibility and personality during service provision, and alleviate the human attention and interaction bottlenecks by providing self-management features using contexts, including self-configuration, self-adaptation, self-optimization, self-protection and self-healing (through self-diagnosis). This is vital to achieve the vision of Ambient Intelligence (AmI) that should come with the pervasive middleware like Hydra (IST-2005-034891).

To facilitate achieving context-awareness, we agree that OWL[1] ontology is the best way for context modeling [2], which can provide reasoning potentials for what contexts we are in, a capability not easily achievable by other context modeling approaches. The definition of *context* in [1] is general enough to cover the contexts in pervasive computing, but we want to point out that not only static knowledge, but also dynamic and runtime context should be considered in order to handle runtime-related requirements. For example, we can run a status check of a system at runtime, and monitor the dynamic contexts of the system and then make decisions on where the problem is, why the problem happens and how to tackle the problem.

[1] OWL homepage. http://www.w3.org/2004/OWL/.

Take a concrete agriculture scenario in the Hydra project:
Bjarne is an agricultural worker at a large pig farm in Denmark. As he checks whether the pigs are provided with the correct amount of food, he is interrupted by a sound from his PDA. Apparently, the ventilation system in the pig stable has malfunctioned. After acknowledging the alarm, the system begins to diagnosis and soon it decides that the cause of the problem is 'power supply off because of fuse blown'. Then he can prepare a fuse and repair the ventilator. After repairing it, he signs off the alarm, and chooses one of the predefined log messages, describing what he has done.

As can be seen from the above scenario, it is very important that the Hydra middleware can provide diagnosis functionality to the end user, or better to achieve self-healing when there is malfunction. To this end, the run time information, for example the device states should be monitored in order to make decisions on diagnosis. Other self-managing work for the Hydra middleware among others includes self-protection according to security rules, searching service and negotiating QoS (Quality of Service) parameters with other services.

In this paper, we will show the Hydra context ontologies that considering run time contexts, and present an OWL ontology and SWRL (Semantic Web Rule Language)[2] based self-management approach for Hydra, in particular the self-diagnosis, which take into the characteristics of pervasive computing. We demonstrate our approach with the Diagnosis Manager for Hydra middleware. The evaluations in terms of extensibility, performance, and scalability shows that the proposed Hydra OWL/SWRL context ontologies and self-management approach based on OWL/SWRL context ontologies are effective to achieve the self-diagnosis goals, and lay a solid foundation for other self-management work.

The rest of the paper is structured as follows: in Section 2 we will briefly introduce the architecture of self-management based on OWL/SWRL ontologies of the Hydra middleware; We then show the Hydra ontologies that facilitate self-management, followed in Section 4 we present the complex context specification with SWRL; In Section 5, we demonstrate the proposed approach with the Diagnosis Manager together with some evaluations. We compare our work with the related work in Section 6. Conclusions and future work end the paper.

2 Architecture of self-management in Hydra

The Hydra project is developing self-managed middleware for pervasive embedded and network systems based on service-oriented architecture. Several components are involved in achieving the self-management features, based on context ontologies where dynamic contexts are encoded. These components include a Diagnosis Manager, which is used to monitor the system conditions and states in order to fulfill error detection, diagnosis, and provide recovery solutions; and a QoS Manager negotiating QoS parameters with other services and manages resources accordingly. Further, the QoS Manager provides device specific infor-

[2] SWRL specification homepage. http://www.w3.org/Submission/SWRL/

mation to the Diagnosis Manager, and coordinates with Service Manager, Ontology Manager and Orchistration Manager. Context events are managed using an Event Manager where publish/subscribe functionalities are provided.

To build necessary intelligence into the Hydra middleware in order to support the self-management, the related monitoring and dianosis rules, QoS rules and service selection rules built on top of the ontologies, play a vital role. We chose SWRL to develop these rules. SWRL is a W3C recommendation for the rule language of the Semantic Web, which can be used to write rules to reason about OWL individuals and to infer new knowledge about those individuals. SWRL provides builtins such as math, string and comparisons that can be used to specify extra contexts, which are not possible or very hard to achieve by OWL itself. Therefore, SWRL is naturally chosen as the rule language in Hydra for the implementation of self-management rules.

The architecture for all the self-management components is the same for Hydra. We are following the three Layered architecture proposed by Kramer and Magee [3]. Based on the current status of OWL/SWRL, we came up with the following architecture as shown in Figure 1, in which the *Goal Management*, *Component Control* and *Change Management* are enclosed with dashed line. The bottom of the architecture is the ontologies/rules, in which knowledge of devices, rule based QoS, and state based diagnosis are encoded. For the Component Control layer, it is mainly used for state reporting and run time information updating, for example battery level and QoS measurement. For the Change Management layer, it is used to response to the state reported from the Component Control layer, and then execute rules developed based on these state and other run time information. For the Goal Management layer, it is used to resolve the rule conflicts based on QoS regulations or user preference etc.

3 Context ontologies in Hydra

Context-awareness, especially the awareness of those dynamic context information, is the most important factor to fulfill the goal of various self-management processes. In Hydra, the context awareness has the following awareness features:

- *Resource awareness* This includes hardware, for example CPU, and software, for example operating system.
- *Power awareness* Different network carriers use different amount of energy during transmission. This will be considered during service provision. Battery information for device is also need to be known.
- *QoS awareness* As one of the important criteria for the selection of service, QoS is another context that shows both static and dynamic affects to the middleware, for example latency.
- *Security awareness* The right information should be transferred to the right user at the right time in the right place using the agreed service level agreement, and in the appropriate format.

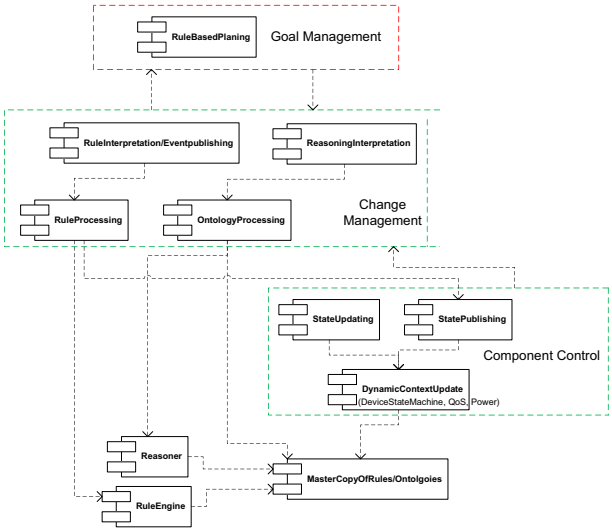

Fig. 1. Architecture of the self-management in Hydra

3.1 Structure and design of the Hydra ontologies

Although there are some pervasive computing ontologies, e.g. SOUPA[3] ontologies, they are not enough for achieving the needed intelligence and handling of dynamic context in order to achieve the above mentioned various awareness. The openness and dynamism of pervasive computing, and the nature for pervasive and embedded devices running as state machines, motivate the development of Hydra context ontologies, whose high level structure is shown in Figure 2.

The Device ontology itself is used to define some basic information of a Hydra device, for example device type classification(e.g. mobile phone, PDA, sensor), device model and manufacturer, and so on. The device type classification in the Device ontology is based mainly on AMIGO project ontologies [4]. To facilitate diagnosis, there is a concept called *HydraSystem* to model a system composed of devices to provide services. And there is a corresponding object property *hasDevice* which has the domain of *HydraSystem* and range as *HydraDevice*. There are also concepts used for the monitoring of web service calls, including SocketProcess, SocketMessage and IPAddress. The *HydraDevice* concept has a data-type property *currentMalfunction* which is used to store the inferred device malfunction diagnosis information at run time and will be exemplified later.

The device Malfunction ontology is used to model knowledge of malfunction and recovery resolutions. We separate the malfunctions into two categories: *Error* (including device totally down) and *Warning* (including function scale-down, and plain warning). There are also two other concepts, *Cause* and *Remedy*, which are used to describe the origin of a malfunction and its resolution.

[3] Semantic Web in Ubicomp. http://pervasive.semanticweb.org/.

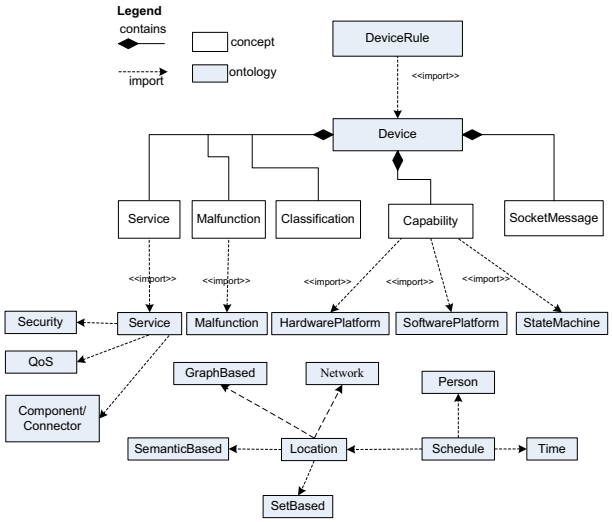

Fig. 2. Structure of the Hydra context ontologies

3.2 Dynamic context

The dynamic context information will reflect the running status of the underlying system, therefore it is the key enabler for the functioning of self-management. For example, the device status is important to check the device working condition and is used to develop diagnosis and monitoring rules; the monitoring of QoS is important to test whether service level agreement is meet due to service mobility.

A common sense of mobile and embedded devices used in pervasive environments is that they are usually designed and operated as state machines. In line with this, a state machine ontology is developed based on [5] with many improvements:

- A data-type property *isCurrent* is used in order to indicate whether a state is current or not.
- A *doActivity* object property is added to the *State* in order to specify the corresponding activity in a state and this makes the state machine complete.
- A data-type property *hasResult* is added to the *Action* (including activity) concept in order to check the execution result at runtime.
- Three data-type properties are added to model historical action results.

The dynamic context is modeled with runtime concepts and properties in the related ontologies, mainly the StateMachine ontology, the Malfunction ontology, QoS ontology, and other concepts and properties in the Device ontology, such as *currentMalfunction* and *HydraSystem*. The *currentMalfunction* will be used to store the current diagnosis information for the malfunction case, *HydraSystem* is used to dynamically model the device joining and leaving and reflect the composition of a system.

Because of the space limit, in this paper we only present the self-diagnosis to demonstrate the effectiveness of the self-management using these ontologies. Figure 3 shows a more detailed but simplified view of the ontologies facilitating diagnosis.

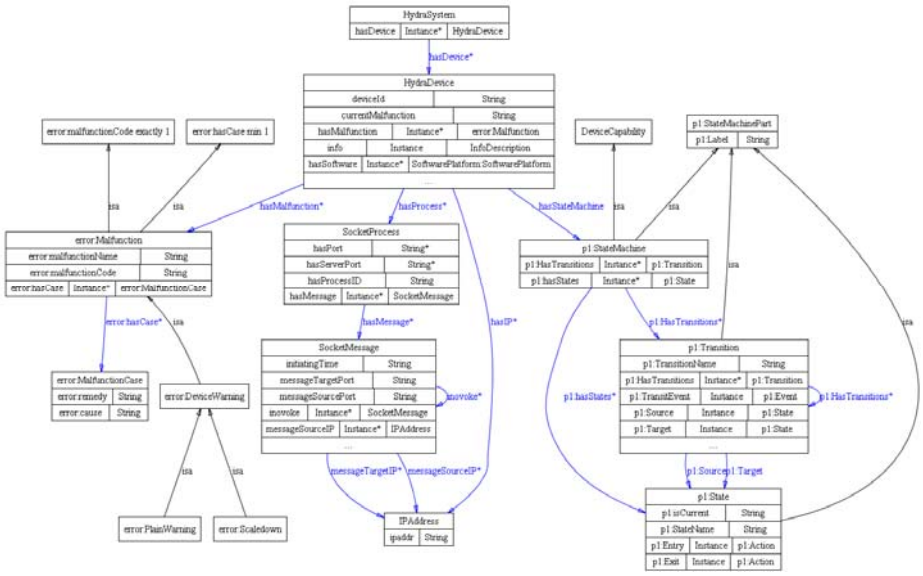

Fig. 3. Partial details of the Diagnosis Manager used ontologies

4 Extending OWL ontologies with SWRL rules

A SWRL rule is composed of an antecedent part (body), and a consequent part (head). Both the body and head consist of positive conjunctions of atoms. A SWRL rule means that if all the atoms in the antecedent (body) are true, then the consequent (head) must also be true. SWRL is built on OWL DL and shares its formal semantics. In our practice, all variables in SWRL rules bind only to known individuals in an ontology in order to develop DL-Safe rules to make them decidable. In our example SWRL rules, the symbol $\wedge$ means conjunction, and $?x$ stands for a variable, $\rightarrow$ means implication, and if there is no ? in the variable, then it is an instance.

4.1 Complex context specification with SWRL rules

SWRL has more expressive power than OWL using various builtins, and can be used to specify complext contexts. As an example, we can specify a GPS distance

calculation with SWRL in order to define a *farAwayFromHome* context (e.g. 5 miles away from home using the GPS distance calculation formula[4]). Then this new context can be used to take actions, for example, surveillance system is switched automatically to the highest security level with all cameras turned on.

$person : hasHome(?person, ?home) \wedge$
$person : inLocation(?person, ?coord1) \wedge$
$loc : hasCoordinates(?home, ?coord2) \wedge$
$coord : latitude(?coord1, ?lan1) \wedge$
$coord : latitude(?coord2, ?lan2) \wedge$
$swrlb : subtract(?sub1, ?lan1, ?lan2) \wedge$
$swrlb : multiply(?squaresublan, ?sub1, ?sub1) \wedge$
$swrlb : multiply(?par1, ?squaresublan, 4774.81) \wedge$
$coord : longitude(?coord1, ?long1) \wedge$
$coord : longitude(?coord2, ?long2) \wedge$
$swrlb : subtract(?sub2, ?long1, ?long2) \wedge$
$swrlb : multiply(?squaresublong, ?sub2, ?sub2) \wedge$
$swrlb : multiply(?par2, ?squaresublong, 2809) \wedge$
$swrlb : add(?parameter, ?par1, ?par2) \wedge$
$swrlm : sqrt(?distance, ?parameter) \wedge$
$swrlb : greaterThan(?distance, 5) \wedge$
$swrlb : StringConcat(?str, '' true'')$
$\rightarrow sqwrl : select(?person, ?home, ?distance) \wedge farAwayFromHome(?person, ?str)$

Similarly, we can define QoS metrics and other QoS regulation with SWRL rules, and we are investigating the specifying of service selection using SWRL rules based on the Security ontology, Service ontology and QoS ontology. Here is a simple example of querying the availability dynamically.

$swrlb : add(?total, p1 : downtime, p1 : uptime) \wedge$
$swrlb : divide(?availability, p1 : uptime, ?total)$
$\rightarrow sqwrl : select(?availability)$

4.2 Complex dynamic context

To achieve self-management, for example self-diagnosis, the OWL context ontologies themselves are really weak to specify rules that are important to define policies for security, QoS metric calculation and diagnosis rules. In these cases, we are applying SWRL to specify these dynamic contexts.

For this paper, we will elaborate on the self-diagnosis contexts which rely on the device state machine and other related concepts as mentioned in the former section. In a similar way, the QoS monitoring rules could be developed based on the dynamic information monitored.

Monitoring and diagnosis rules are the basis for the diagnosis service. We have two level diagnosis rules, namely device level rules and system level rules. Device level rules are used for a certain type of devices which are supposed to be

[4] How to calculate the distance between two points on the Earth. http://www.meridianworlddata.com/Distance-Calculation.asp

generic for that type of devices. The following example rule specifies the mobile phone battery level monitoring. If the battery level is less than 10%, then a warning will be published.

$device : MobilePhone(?device) \wedge$
$device : hasHardware(?device, ?hardware) \wedge$
$Hardware : primaryBattery(?hardware, ?battery) \wedge$
$Hardware : batteryLevel(?battery, ?level) \wedge$
$swrlb : lessThanOrEqual(?level, 0.1)$
$\rightarrow VeryLowBattery(?device)$

System level rules are used to specify rules that span multiple devices in a system. For example, if the detected flow for feeding the pig is more than 6 gallon per minute, then we can infer that the pipe is broken.

$device : FlowMeter(?device) \wedge$
$device : hasStateMachine(?device, ?statemachine) \wedge$
$statemachine : hasStates(?statemachine, ?state) \wedge$
$statemachine : doActivity(?state, ?action) \wedge$
$statemachine : actionResult(?action, ?result) \wedge$
$abox : isNumeric(?result) \wedge$
$swrlb : greaterThan(?result, 6.0) \rightarrow device : currentMalfunction(device : Flowmeter, "PipeBroken")$

A more complex rule is the example of pig farm ventilating and monitoring system, thermometers are used to measure both indoor and outdoor temperature. In the summer time, the indoor temperate should follow the same trend as the outdoor temperature. A rule is developed to first obtain the trends by the difference of continuous temperature measurements of both the indoor and outdoor temperatures. If the trend is not the same, we infer that the ventilator is down.

From our experiences, the loading of the OWL/SWRL ontologies is the main performance bottleneck, therefore all the current rule sets are stored in one separate ontology called *DeviceRule* as can be seen from Figure 2, and we load it when the system initializes.

5 Evaluating OWL/SWRL context ontologies based self-management with Diagnosis Manager

To evaluate the OWL/SWRL context ontologies based self-management approach, and the proposed Hydra ontologies, we will use the Diagnosis Manager as an example to show the effectiveness and usability, in terms of extensability, performance (as the main concern), and scalability.

5.1 Design and implementation of the Diagnosis Manager

Hydra implements a service-oriented architecture based on web service interaction among devices. Initially we focus on device status reporting using state

changes as events through the Hydra Event Manager, and Web service request/reply reporting using IPSniffer tool. When there are state change events, the device state machine instance in the state machine ontology need to be updated, and also these state changes will be published with state machine state changes as event topic. The Diagnosis Manager is an event subscriber to the state machine state change events, it will then update the corresponding state instances in the ontology. At the same time, this will trigger the diagnosis of the device status, executing the SWRL rules to monitor the health status of devices, and also trigger the reasoning of possible device errors and their resolutions. The Diagnosis Manager will publish the diagnosis results as an event publisher.

We adopted a mix of the Blackboard architecture style and the Layered architecture in the actual implementation due to the high overhead for loading ontologies, and use the observer pattern in both the updating of state machine ontology and parsing fo the inferred result from SWRL rules.

OWL ontology provides intelligence capabilities for diagnosis decisions. For example, Bjarne get a high priorit warning of "GrundfosPumpMQ345 failed to start". A diagnosis task is initiated to check what is wrong with the pump, but as a newly installed pump, there is still no error resolution to this model of pump in the Malfunction ontology. As a further step, the diagnosis system will conduct subsumption reasoning and search for the device *Type* in the Device ontology, which is found as *FluidPump*, and then its manufacturer is also queried. Now another query to the Device ontology will get a similar pump called *GrundfosPumpMQ335* as of the same type from the same manufacturer "Grundfos". And based on the name of the error and pump type, the solution from a query to Malfunction ontology is suggested "replace a capacitor", which is happily the solution to solve the problem.

5.2 Evaluation of the Diagnosis Manager

We evaluated the extensibility of the OWL/SWRL based diagnosis Manager in terms of scalability, performance, and extensibility. We started the development of Diagnosis Manager with the rule for temperature monitoring. After finishing the implementation and testing, we then try to handle the flowmeter diagnosis rules. We only need to add the flowmeter rules to the existing rules set. No single line of Diagnosis Manager code needs to be changed. In summary, the Diagnosis Manager has good extensibility.

For the measurement of performance, the following software platform is used: Protege 3.4 Build 125, JVM 1.6.02-b06, Heap memory is 266M, Windows Vista. The hardware platform is: Thinkpad T60 Core2Duo 2G CPU, 7200rpm hard disk, 2G DDR2 RAM. The time measurement is in millisecond. The size of DeviceRule ontology is 210,394 bytes, and contains 22 rules, which is fair for a small pervasive system in which monitoring and diagnosis functions are all included. The performance figures are shown in Table 1. An interesting thing is after some time of running, the Diagnosis Manager is running stably with the total time in 260-270 ms for processing an event, a bit faster than the one when

it starts. Here the parsing of the inferred result is running in a multi-threaded way in the Diagnosis Manager.

Update	InferringTime	AfterEventTillInferred
383	380	382
322	319	321
282	278	282
272	269	271
265	263	265
270	267	269
268	266	269

Table 1. Diagnosis Manager performance

For scalability, a number of events are published (almost in parallel) to measure how long it will be, starting from the publishing till the end of inferring and publish related inferring result. Time needed (y-axis) is shown in Figure 4 (x-axis shows the number of events) . We can see that the time taken is in linear with the events need to be processed.

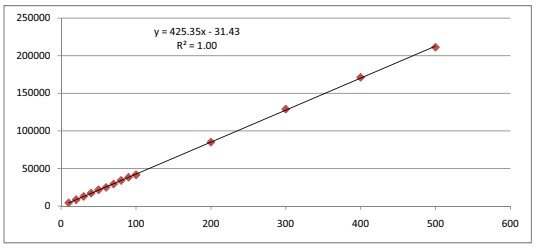

Fig. 4. Diagnosis Manager scalability

6 Related work

The work on *Context OWL*[6] considers that contexts are *local*. In the pervasive computing environment, this is not always true as we have a global *Time* manager which manages the consistent time for all users. Work in [7] also applied SWRL-based context modeling, and illustrated three cases of applying SWRL to manipulate context. We go beyond this work by the dynamic-state based monitoring and diagnosis using the context ontologies. When compared to the existing pervasive computing ontologies, such as SOUPA and Amigo[4], the Hydra context ontologies have some unique features. Firstly, dynamic contexts are incorporated which facilitate the achieving of self-management. Secondly, complex contexts, especially those self-management needed complex and dynamic

contexts, are defined by SWRL, which are not expressible by OWL ontology itself.

Kramer and Magee [3] recently proposed a reference model for self-managed systems, which is composed of component control, change management and goal management. In this paper, we largely followed this work for the Layered architecture, but mainly focus on the Component Control and Change Management. At the same time, a mix of Blackboard architecture and Layered architecture are applied to improve performance and extensibility.

Self-healing is one of the main challenges to autonomic pervasive computing. Generally speaking, our approach applied the same idea of ETS [8], in terms of the using of states for detecting source of failure, and then notification of failure source. And this process is actually universal for error detection. Our ontology and SWRL rule based approach provides a way of intelligent detection and resolution, which is not easily achievable by ETS.

Work in [9] also use semantic web approach for achieving self-managing. Our approach is non-intrusive, SWRL rules are automatically executed using state machine instead of explicitly inserting sensor code to program, and is more suitable for the characteristics of pervasive devices.

Various failures in a pervasive system are classified in [10], and an architecture for fault tolerant pervasive computing is proposed. We focus not only on device failure monitoring, but also on system level detection using the relationships of different state machine instances. In addition, our approach can be more intelligent in terms that ontology reasoning can help the diagnosis.

There are many researches dealing with the diagnosis in various field, e.g. [11] from traditional artificial intelligence point of view. These traditional approaches are not utilizing the context ontologies that are already there in pervasive systems and are used for context-awareness and other purposes. In our vision, the open world assumption in OWL/SWRL, and hence in our approach, is very well suited for the openness of the pervasive computing environment, which automatically rejects the approaches using Prolog kind of rules that use close world assumption.

7 Conclusions and future work

Self-management capabilities are important to achieve necessary dependability in pervasive system, and is a challenge for pervasive computing. In Hydra, we make use of the OWL/SWRL ontologies as the basis for the implementation of self-management features, which is very suitable for the openness nature of pervasive computing. These context ontologies are incorporating the dynamic context information, including device and service run time information, which can then be used for running status checking and diagnosis, QoS monitoring, and further to achieve other self-management features, such as the self-configuration and self-adaptation.

We illustrate the OWL/SWRL based self-management with the experiment of the Diagnosis Manager, mainly using state machine ontology and SWRL rules

built based on it. The malfunction information and its resolution are encoded in an OWL ontology, and can be used at run time to infer the solution to the malfunction, and further to fulfill self-healing activities. SWRL is used to develop monitoring and diagnosis rules, which can help make intelligent decisions when there is malfunction occurs. The evaluations of the Diagnosis Manager in terms of extensibility, scalability, and performance, relieved us for the worrying of performance of the OWL/SWRL based self-management.

In the future, we will continue the implementation of the Goal and planning layer of the three layered architecture. At the same time we are working on QoS ontology rules and QoS-awareness service matching and service selection based on the SWRL rules. Further work on full scope of self-management, such as self-adaptation, self-configuration based on OWL/SWRL ontologies are under way, which will be reported in the coming papers.

Acknowledgements

The research reported in this paper has been supported by the Hydra EU project (IST-2005-034891).

References

1. Dey, A.: Understanding and Using Context. Personal and Ubiquitous Computing **5**(1) (2001) 4–7
2. Strang, T., Linnhoff-Popien, C.: A Context Modeling Survey. Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp (2004) 34–41
3. Kramer, J., Magee, J.: Self-Managed Systems: an Architectural Challenge. International Conference on Software Engineering (2007) 259–268
4. IST Amigo Project: Amigo middleware core: Prototype implementation and documentation, deliverable 3.2. Technical report, IST-2004-004182 (2006)
5. Dolog, P.: Model-driven navigation design for semantic web applications with the uml-guide. Engineering Advanced Web Applications, In Maristella Matera and Sara Comai (eds.) (Dec. 2004)
6. Bouquet, P., Giunchiglia, F., van Harmelen, F.e.a.: C-OWL: Contextualizing ontologies. Second International Semantic Web Conference (2003) 164–179
7. Plas, D.J., Verheijen, M., Zwaal, H., Hutschemaekers, M.: Manipulating context information with swrl. I/RS/2005/117, Freeband/A-MUSE/D3.12 (2006)
8. Ahmed, S., Sharmin, M., Ahamed, S.: ETS (Efficient, Transparent, and Secured) Self-healing Service for Pervasive Computing Applications. International Journal of Network Security **4**(3) (2007) 271–281
9. Haydarlou, A.R., Oey, M.A., Overeinder, B.J., Brazier, F.M.T.: Use-case driven approach to self-monitoring in autonomic systems. The Third International Conference on Autonomic and Autonomous Systems (2007)
10. Chetan, S., Ranganathan, A., Campbell, R.: Towards fault tolerant pervasive computing. Technology and Society Magazine, IEEE **24**(1) (2005) 38–44
11. Barco, R., Díez, L., Wille, V., Lázaro, P.: Automatic diagnosis of mobile communication networks under imprecise parameters. Expert Systems With Applications (2007)