Wapee: A Fault-Tolerant Semantic Middleware in Ubiquitous Computing Environments

Yoonhee Kim[1], Eun-kyung Kim[1], Beom-Jun Jeon[2], In-Young Ko[2], and Sung-Yong Park[3]

[1] Dept. of Computer Science, Sookmyung Women's University, Seoul Korea
{yulan, kimek}@sookmyung.ac.kr
[2] School of Engineering, Information and Communications University, Korea
{shadow, iko}@icu.ac.kr
[3] Department of Computer Science and Engineering Sogang University, Seoul Korea
parksy@sogang.ac.kr

Abstract. A middleware in ubiquitous computing environment (UbiComp) is required to support seamless on-demand services over diverse resource situations in order to meet various user requirements [11]. Since UbiComp applications need situation-aware middleware services in this environment. In this paper, we propose a semantic middleware architecture to detect errors, analyze causes of errors, and plan semantically meaningful strategies to deal with a problem with associating fault and service ontology in UbiComp environment. We implemented a referenced prototype, Web-service based Application Execution Environment (Wapee), as a proof-of-concept, and showed the efficiency in runtime recovery.

Keywords: Ubiquitous, semantic, ontology, fault-tolerance.

1 Introduction

The advent of Ubiquitous Computing (UbiComp), which runs dynamically over heterogeneous environment emphasizes the needs of service-oriented middleware services in the concept of computing anytime, anywhere, and any devices, instead of resource in computing environment. In the UbiComp environment, the concept of situation-aware middleware has played an important role in meeting user needs with available computing resources appropriately in dynamic environment. An UbiComp system consists of a heterogeneous set of computing devices; a set of supported tasks; and some infrastructures the devices may rely on in order to carry out their tasks. It hides the heterogeneity of the resource environments and provides necessary services to UbiComp applications.[1]

As the diversity and complexity of situations in UbiComp environment, it is not trivial and realistic to come up with semantically meaningful middleware services to support high availability, especially to recover from faulty situations with predefined recovery strategies in real world. In addition, pursing sophisticated controls over complicated faulty situation takes quite amount of time to analyze the cause and plan

[1] This work was supported by the SRC/ERC program of MOST/KOSEF (R11-2005-017).

X. Zhou et al. (Eds.): EUC Workshops 2006, LNCS 4097, pp. 173 – 182, 2006.

recovery strategies to support fault tolerance, in order to achieve service continuity in various running environment.

Fault-tolerance issues have been addressed in various areas of computing systems such as computer architecture, operating systems, distributed systems, mobile computing and computer networks. In this paper, we discuss semantically meaningful fault-tolerant middleware architecture to improve availability of application services in UbiComp environments. In this paper, we have suggested a semantic middleware architecture for fault tolerance with application fault ontology to provide high availability service delivery. To enable a service to seamlessly run in ubiquitous environment, we introduce the Web-service based Application Execution Environment (Wapee). It consists with Fault Management (FM) and Runtime Service Management (RSM) with high fault-tolerance, or continuous availability. The FM provides ontology-based context understanding service in the application areas. The RSM can be dynamically service reconfiguration by the runtime service manager. Both are presented for the fast execution time, fault-tolerance and continuous availability.

The rest of paper is organized as follows. The related works are introduced in section 2. Section 3 presents overall architecture and the detailed description of Wapee. In section 4, the experiments of our prototype have demonstrated the semantically meaningful fault detection and recovery functionality of the mechanism in our architecture and the efficiency in runtime. We conclude with some directions for future work at the end of this paper.

2 Related Works

Research on fault tolerance has been more emphasized to provide seamless and continuous services in Grid, ubiquitous, or distributed computing environment.

Grid Enactor and Management Service (GEMS) [4] supports the detection of individual job process failures for parallel message-passing applications. Failed Jobs can be canceled and restarted, either on the same local resource if sufficient nodes are available in a restart queue, or on another resource. GEMS requires that a local resource manager support certain fault-detection and reporting capabilities.

CORBA [2] have long lacked real support for fault tolerance. In most cases, a failure was simply reported to the client and the system undertook no further action. For example, if a referenced object could not be reached because its associated server was unavailable, a client was left on its own. In CORBA version 2.6, fault tolerance is explicitly addressed.

The Adaptive Reconfigurable Mobile Objects of Reliability (Armor) [3] middleware architecture offers a scalable low-overhead way to provide high-dependability services to applications. It uses coordinated multithreaded processes to manage redundant resources across interconnected nodes, detect errors in user applications and infrastructural components, and provide failure recovery. The authors describe their experiences and lessons learned in deploying Armor in several diverse fields.

3 Wapee Overview

Wapee (Web-service based Application Execution Environment) focuses on providing autonomic fault-tolerance services with fault detection, fault analysis and recovery with application level service reconfiguration and its runtime level deployment (see Figure 1). Application level service reconfiguration can be achieved by autonomic detection and analysis services in application level Fault Management with semantically meaningful ontology of U-services and faults in a ubiquitous environment. The service reconfiguration information in an Application Description Graph (ADG) is fed in to Runtime Service Management (RSM) to be realized as U-services on a prepared resource pool. Based on the ADG, the RSM asks RSL Generation Service to create Application Deployment Description (ADD), which includes service deployment information such as resource description of service managers, local schedulers, input and output data file path, and executables; and runtime dependency of the U-services in the ADG.

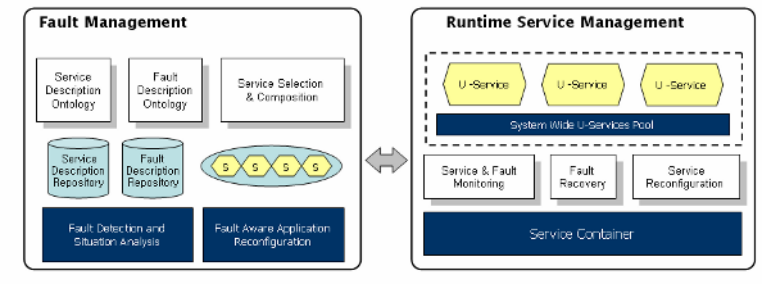


Fig. 1. Architecture of Wapee

3.1 Fault Management

When a fault cannot be resolved in the service manager level, the Wapee's fault manager reconfigures the application to utilize an alternative service that provides the same or similar functionality as the service that caused the fault. There are some requirements of the application-level fault manager to ensure the functional reliability and continuity of an application:

- *Functional consistency*: An alternative service must provide the same or similar functionality as the original one to achieve the consistent goal.
- *Interoperability*: An alternative service must be interoperable with the adjacent services of the original one. Not only the interface-level interoperability, but also the semantic interoperability among the adjacent services must be ensured.
- *Effectiveness*: An alternative service must be selected in a way that the service contributes to resolve the fault situation.
- *Operational continuity*: The execution of an application must be continued after the reconfiguration of the application structure with an alternative service.

To meet these requirements, the fault manger in our framework supports description models to formally describe the types of fault conditions and the functionality of services. The fault manager also provides a service brokering mechanism that identifies a fault condition based on an exception event and service status, and finds alternative services that are interoperable with other services in an application and effectively resolve the fault condition.

3.1.1 Ontology-Based Fault and Service Description Models

We have developed ontology-based description models to describe semantics of service faults and functionalities. We define three ontology hierarchies: the fault, service, and recovery strategy ontologies. The fault ontology is for abstracting types of faults based on their causes such as the limitation of memory resource, and service errors. The fault ontology has a property to represent the resource condition that might cause a fault. The service ontology is for describing the functionality and resource requirements of a service. Finally, the recovery strategy ontology is for describing possible strategies to resolve a fault condition.

The fault ontology includes a property that holds a pointer to a recovery strategy that might resolve the fault condition. The candidate services that can substitute the faulty service are dynamically inferred based on the strategy specified in this ontology. This tri-structure ontology makes the service brokering task much more flexible and scalable by allowing faults and services not to be directly associated and separately managed.

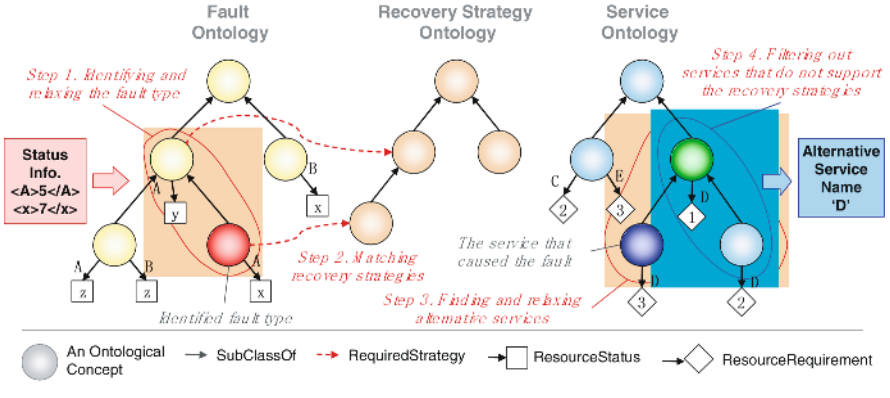


Fig. 2. Major steps of the semantically-based service brokering process

3.1.2 Semantically-Based Service Brokering

Fig. 2 shows the major steps to find alternative services of a service that caused an exception. When an exception occurs in a service, the system reports the current status of the service and its environment. The service broker matches this fault information against the resource-condition property of the fault ontology to identify the corresponding fault semantics [6]. To find relevant fault semantics as much as possible, we adopt a semantic relaxation method, which, in an ontology hierarchy, collects nodes that have the same set of properties and are on the same subsumption hierarchy – direct parents and children (Step 1 in Fig. 2).

Once a set of possible faults is identified, the service broker retrieves relevant recovery strategies to resolve the faults (Step 2 in Fig. 2). The service broker then finds services that provide the same or similar functionality as the original service. A semantic relaxation method, which is similar to the method that we used for the fault ontology, is applied to the service ontology to extend the service set (Step 3 in Fig. 2). The resource-requirement property of each service is then compared with the resource description in each of the recovery strategies retrieved. Only the services that can contribute to resolve the fault (the services that meet the resource requirements) are selected as candidate services that can be used to substitute the original service (Step 4 in Fig. 2).

3.2 Runtime Service Management (RSM)

RSM is responsible for job execution management and interaction with users (See Fig. 3). The RSM make estimates of the resource usage of job submissions in order to ensure efficient use of grid resources [1]. Examples of service failures include service crashes due to bugs and operating system errors, faulty operation of services like sensing incorrect context, wrong inferring delivery of events. Service failures can potentially lead to failure of the UbiComp system.

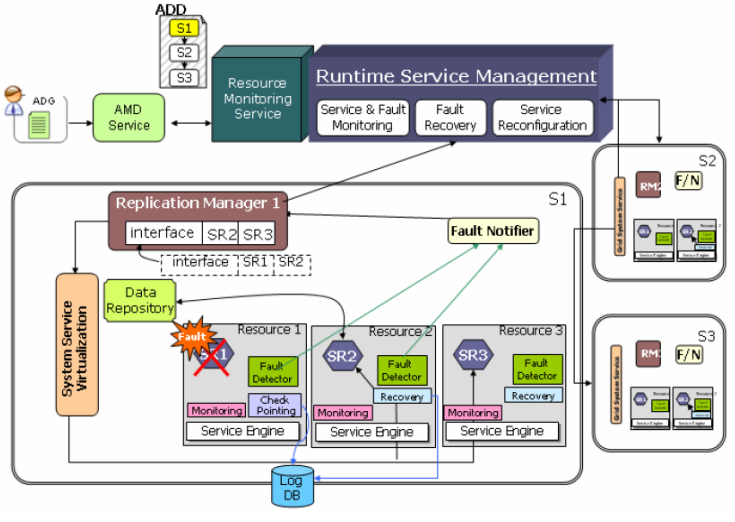


Fig. 3. The architecture of RSM

3.2.1 Job Submission Service

One of main services a runtime application in Ubiquitous Environment must provide is to job submission to remote resources. On such environment, users can execute jobs that consist of large number of independent tasks with a single sign-on authentication. We are able to support such uniform job submission to remote computing resources while using the Grid Resource Allocation and Management (GRAM) in Globus [7] toolkit to access Grid resources securely.

The client creates a request file by using the Wapee Application Client. The component of submission service will create a job description file using the XML.

This file includes details of which distributed machine will be used, where the data files are, and where the result file should be written. Then, the job submission component will invoke the WS-GRAM service on the remote computation resource with the XML file. The WS-GRAM resource on the remote site will parse the XML file and submit a job to the local job scheduling system[5].

3.2.2 Monitoring Service

The purpose of Monitoring service is to provide real-time job monitoring and status feedback to a steering service while operating in close interaction with an execution service, such as Condor, to provide interactivity, fault tolerance and error detection. Once a job is submitted in Wapee, Monitoring services periodically monitors a job that has been submitted for execution in the Virtual Organization (VO) and reports job status. A VO is basically groups that are authorized to run Grid jobs on a set of Grid resources. Whenever the state of a job changes the Monitoring service will update the repository. WS-GRAM [10] supports querying job status and monitoring of output and error streams of running jobs. It will interact with execution service to collect monitoring data and then this data will be stored in the data store. Monitoring data will be provided to the clients once it has been requested.

3.2.3 Replication and Service Reconfiguration

To meet the requirement of high availability and fault tolerance, replication scheme is used. Fig. 3 depicts the implementation of the RM in a typical deployment scenario at a local site replicates data from one or more remote sites. The operations of RM include location, identifying where desired data files exist on the Grid; transfer, moving the desired data files to the local system efficiently; and registration. We considered primary-backup replication for achieving fault-tolerance.

It also addresses automatic reconfiguration because different invocations of the same service may result in the selection of different components. In the Wapee architecture, the RSM is primarily responsible for planning and initiating configuration changes in the system. Development of this adaptive reconfiguration mechanism requires identification of output information provided by the system and input information that the mechanism can inject into the system to affect change. The dynamic resource management service we have designed is in charge of detecting configuration changes, updating the distribution of directory entries on cluster nodes in the event of a configuration change, triggering reconfiguration of distributed services when needed.

4 Wapee Implementation

Our main goal is to develop a workflow solution for complex grid applications to support the design, execution, monitoring, and performance visualization phases of development in a user-friendly way. We have developed a GUI based tool, Wapee Client, for workflow management, as shown in Fig. 4. A visual interface that allows for the graphical manipulation of workflow process instances provides a rich medium for the communication of dependencies and relationships between constituent jobs of a workflow process instance.

A job in workflow is represented by a set of interdependent tasks arranged in a Directed Acyclic Graph (DAG) [9]. After the creation of the DAG the resources identified in the workflow must be mapped onto the available grid resources [8]. The RSM supports run-time execution and job monitoring. Output results can also be available for a view from the Wapee Client.

5 Experiment Results

An experiment scenario is web-based applications, such as aggregation, searching and ranking about enormous web-based information. First, user can gather tremendous editorials on various newspaper website in the same breath using 'Wrapper Applications' of distinct type. Each 'Wrapper Application' takes different time when it finishes. We choose three 'Wrapper Applications' for this experiment. And then, user can both view the result and send input-file for other applications at next phase. We select 'Ranking Application' and 'Search Application' for mid-applications of our experiment. The 'Search application' searches some words at forepart result. The 'Ranking Application' finds selected word at forepart result and then shows ranking. Finally we join the whole information through different applications using 'Aggregation Application'.

For example, when a fault occurs at 'Searching Application' phase, Wapee analyze fault properties and classify the fault type, and replace another useful similar 'Searching Application' using fault recovery strategy of Fault Manager. On Fig. 5, we show a workflow of our scenario.

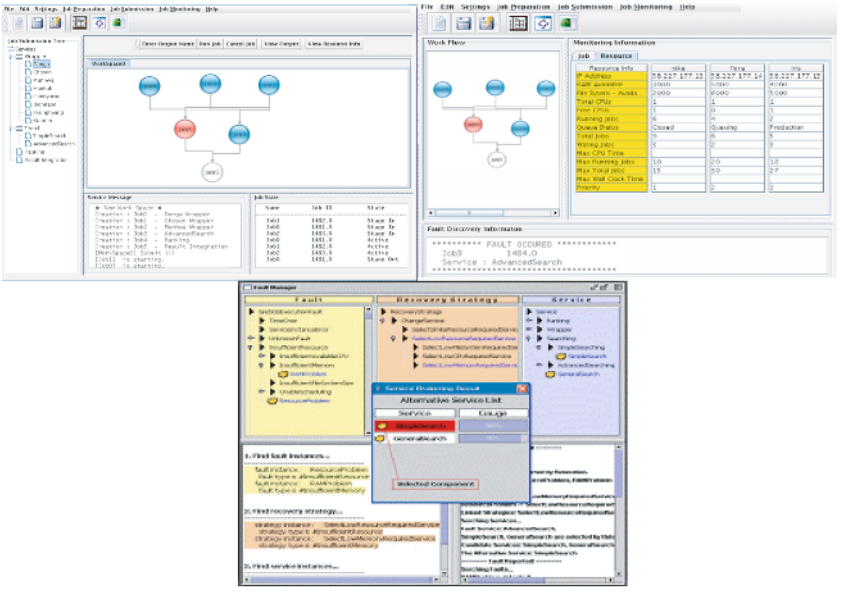


Fig. 4. Implementation of prototype

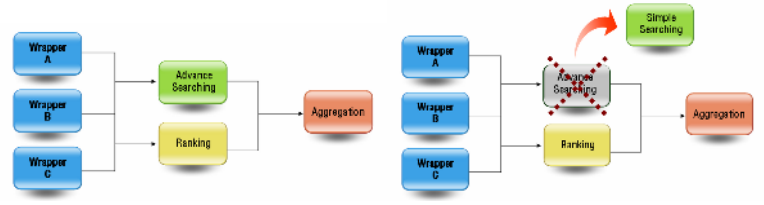
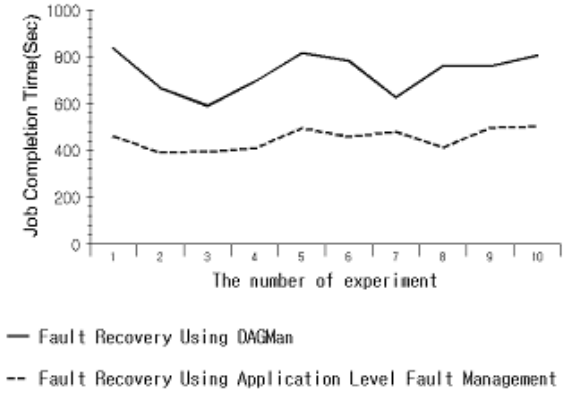
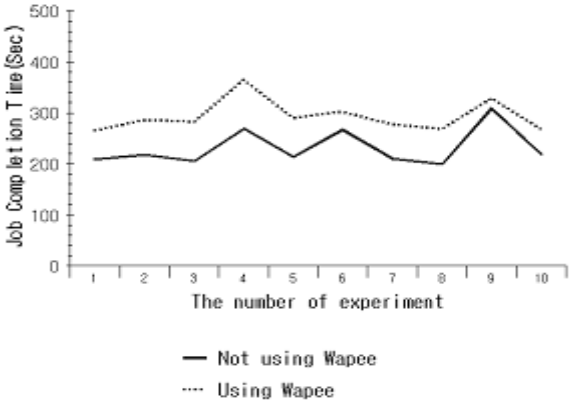


Fig. 5. Our test scenario: If fault occurs during using 'Advance Searching' application, we can overcome the fault using RSM and FM. If fault is classified that cannot be resolved at the runtime service manager level. To overcome such situation, we extend the fault handling mechanism to the application level, Fault Manager, such that the application can be reconfigured to utilize an alternative service that provides the same or similar functionality as the service that caused the fault. Its case is alterative service, 'Simple Searching'.



(a)



(b)

Fig. 6. Experiment results

In Fig. 6 we showed the success rate and percentage of used fault-tolerance mechanism in Wapee. If Runtime-Level fault occurs, as shown in Fig. 6 (a), Wapee detect fault and recover them through Runtime Service Manager (RSM). The whole procedure takes about 326 seconds. This fault-tolerance mechanism is very basic algorithms that try to allocate resources on the nearest surrogate possible.

If faults cannot be resolved at the service manager level then the RSM notify the fault handling information to the Fault Manager at application level. The whole procedure takes about 350 seconds, if Wapee detected these faults and recovered them using semantically Ontology, as shown in Fig. 6 (b).

These figures tell us that using fault recovery system, Wapee, increases service availability and executes resource efficiently in ubiquitous computing environments. It also shows us that the overhead ratio of middleware and application is kept in a relatively stable level (16.16% using RSM, 24.68% using FM) regardless of the variation of resource environment and service configurations. Our experiment validates the practicability and soundness of Wapee. The overhead of middleware is kept in a small ratio with respect to the overall system cost.

6 Conclusion and Future Works

Wapee, a fault-tolerant semantic middleware, executes likely faulty applications successfully with semantically meaningful service and fault ontology in ubiquitous computing environments. When a fault is found in runtime execution, Runtime Service Management (RSM) autonomically identifies the faults and decides if the fault might be resolved in runtime level or not. For resolvable faults in runtime, RSM configures Application Deployment Description again to obtain alternative resources for the application. Otherwise, Fault Management reconstructs alternative Application Description Graph (ADG) with the help of the semantics of services and faults ontology; and informs the ADG for new deployment of the application autonomically. In addition, Wapee client, one of other strengths of Wapee, provides easy-of-use user interface for application construction, runtime execution, real-time monitoring and visualization of results.

For future work in Wapee, we are planning to implant an effective and autonomic meta-scheduler in collaboration with various local schedulers. Scheduling will be done with some consideration of application configuration information, environmental condition, user profile, and other special requirement such as fault tolerance policies to improve the quality of an application and resource utilization.

References

1. I. Foster. C. Kesselman, S. Tuecke. “The Anatomy of the Grid: Enabling Scalable Virtual Organizations” International J. Supercomputer Applications, 2001.
2. CORBA Fault http://www.omg.org/cgi-bin/apps/doc?formal/01-09-29.pdf
3. Zbigniew Kalbarczyk,Ravishankar K Iyer, Long Wang," Application Fault Tolerance with Armor Middleware" Internet Computing, March/April 2005 (Vol 9, No 2) pp 28-37
4. Satish Tadepalli, Calvin Ribbens, Srinid Varadarahan "GEMS: A Job Management System for Fault Tolerant Grid Computing", High Peformance Computing Symposium, 2004

5. Matthew L Massie, Brent N Chun, David E Culler,"The Ganglia Distributed Monitoring System: Design, Implementation, and Experience",Parallel Computing, Vol 30, Issue 7, July 2004
6. Y. Hainning, E. Letha, “Towards a semantic-based approach for software reusable component classification and Retrieval”, In Proceedings of the 42nd annual Southeast regional conference, 110-115, 2004
7. Globus Project, http://www.globus.org/
8. James Frey, Todd Tannenbaum, Ian Foster, Miron Livny, and Steven Tuecke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids", Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC10) San Francisco, California, August 7-9, 2001
9. Condor DAGMan http://www.cs.wisc.edu/condor/dagman/
10. K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman "Grid Information Services for Distributed Resource Sharing", Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), IEEE Press, August 2001
11. M. Weiser, "The computer for the 21st Century." Scientific American, Vol. 265, No. 3, pp. 94-104, September, 1991