Ontology based Software Reconfiguration in a Ubiquitous Computing Environment

Yoonhee Kim[1], Eun-kyung Kim[1], Jeuyoung Kim[1],Eunhye Song[1], In-Young Ko[2]
[1]*A Dept. of Computer Science, Sookmyung Women's University, Korea*
{yulan, kimek, wldud5, grace}@sookmyung.ac.kr
[2]*School of Engineering, Information and Communications University, Korea*
iko@icu.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 [1]. Since UbiComp applications need situation-aware middleware services in this environment. In this paper, we propose a semantic middleware architecture to support dynamic software component reconfiguration based fault and service ontology to provide fault-tolerance in a ubiquitous computing environment. Our middleware includes autonomic management to detect faults, analyze causes of them, and plan semantically meaningful strategies to deal with a problem with associating fault and service ontology trees. We implemented a referenced prototype, Web-service based Application Execution Environment (Wapee), as a proof-of-concept, and showed the efficiency in runtime recovery.

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.

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 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 to support dynamic software component reconfiguration based fault and service ontology to provide fault-tolerance in a ubiquitous computing environment. 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

* This research was supported by the MIC(Ministry of Information and Communication), Korea, under the ITRC(Information Technology Research Center) support program supervised by the IITA(Institute of Information Technology Assessment)

Proceedings of The Sixth IEEE International Conference on Computer and Information Technology (CIT'06)
0-7695-2687-X/06 $20.00 © 2006 IEEE

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 and 5, 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[2], ubiquitous, or distributed computing environment.

Grid Enactor and Management Service (GEMS) [3] 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 [4] 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) [5] 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) is a middleware for UbiComp environments contrived with the aim of supporting an application to configure and adapt itself to the underlying environments. A key in this architecture is on how resource and service management are faced, and what is the resulting abstraction to the user.

We expect that there will be so many similar service instances in UbiComp environments. However, we can not decide which service instance is most relevant to the application's current situation. For selecting most appropriate one among multiple similar service instances, service discovery should be aware of context information of applications. Service instances are evaluated based on the extent of fitness to current context such as current location, or preferences. To evaluate service fitness, Wapee 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 Fig. 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 Autonomic Management Generation (AMD) 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.

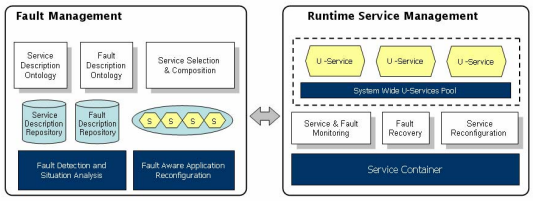


Figure 1. Architecture of Wapee

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

Proceedings of The Sixth IEEE International Conference on Computer and Information Technology (CIT'06)
0-7695-2687-X/06 $20.00 © 2006 IEEE

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.

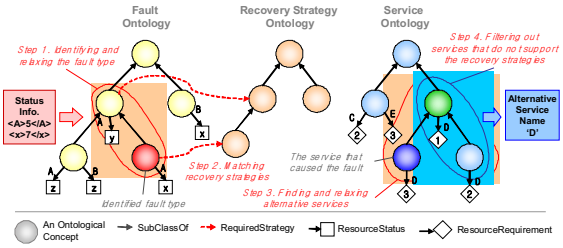


Figure 2. Major steps of the semantically-based service brokering process

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 instantiating and monitoring service (See Fig. 3). The RSM makes estimates of the resource usage of job submissions in order to ensure efficient use of grid resources [8]. 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.

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. Whenever the state of a job changes the Monitoring service will update the repository. It supports querying job status and monitoring of output and error streams of running jobs. Resource Monitoring Service gathers information of resource in VO.

The RSM also addresses autonomic reconfiguration because different invocations of the same service may result in the selection of different components. In the Wapee architecture, it 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.

Autonomic Service Reconfiguration interacts with other components of RSM or Fault Management to search currently available services to be suitable to the context change. To adopt new service, it should check and verify available resources or resource conflicts among services to avoid service crush or malfunction of applications.

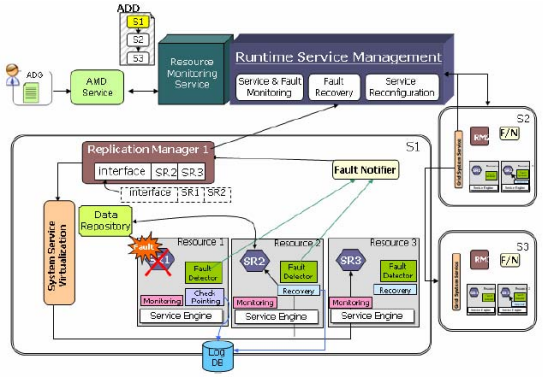


Figure 3. The architecture of RSM

To meet the requirement of high availability and fault tolerance, replication scheme is used. Fig. 3 depicts the implementation of the Replication Manager (RM) in a typical deployment scenario at a local site replicates data from one or more remote sites. The RM replicates the data and the processes of an application, and distributes the replicas across the processors in the system. The Fault Detector of Wapee offers their fault notifications to the replication manager, thereby allowing it to restore the degree of replication if a replica has crashed. When a fault occurs, Fault Detectors detect fault in the objects, and report faults to the Fault Notifier. The Fault Notifier receives reports from the Fault Detectors, and propagates the reports as fault event notifications. The Runtime Service Management reasons about the fault reports that it has received. 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.

3.2.1. Fault-tolerant Approaches. Our middleware makes context aware applications easy to be developed and deployed. When context change from an application is acquired and the change is resulted as a fault, the middleware reconfigures Application Deployment Description to meet requirements with the help of Fault Management(FM). The Runtime Service Management (RSM) has a reconfigured ADD. To create reconfigured ADD, RSM requests FM to identify faults and provide recovery strategy using fault, service, and recovery strategies ontology trees. The reconfigured services that passed through FM's service brokering process are notified to RSM to generate ADD. When the reconfigured context in ADD is activated, user level application functionalities will be provided continuously. The following procedure is Autonomic management for fault-tolerance. A fault tolerance process is summarized in the following pseudo-code (Table. 1).

Table 1 pseudo-code

```
program
  begin
    // Application and domain
requirements from ADG
    set Application Requirement
     {job_ID, job_type, executable,
serviceName,
      arguments,library_path}
    set Domain
     {max_time, max_cpu_time,
max_memory, min_memory}
    Service_Configuration{}
    Service_Instantiation{}

    Monitoring_of_Resources {return
Resource_Info}
    Execution {
      Job Running (event
Context_Change);
        if (Context_Change == fault) {
          Fault_Categorizes();
          if(Fault_Level == Runtime) {
            AlterResource =
Monitoring_of_Resources();

Service_Instantiation(AlterResource);
          }else {
            Monitoring_of_Resources();
            AlterService =
Fault_Management ();

ServiceConfiguration(AlterService);
          }
        }
    Monitoring_of_Jobs { return
Context_Change;}
end.
```

User can create the ADG through setting of application and domain. And then ADD is configured and service is initiated. When a fault occurs during execution, an autonomic management will be executed by the RSM and FM with fault properties. If a fault is classified that can be resolved at the runtime service level then it takes only service re-instantiation. In other case, we extend the fault handling mechanism to the application level, to FM, such that services can be

reconfigured to utilize alternative services that provide the same or similar functionality as the service that caused the fault. Because of autonomic fault tolerance, a system maintains its level of reliability and availability, through reconfiguration in response to changes in its environment of execution.

4. Implementation

A prototype 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.

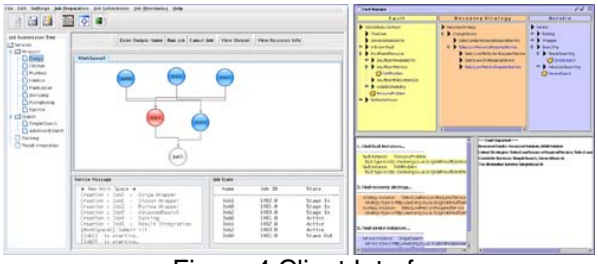


Figure 4 Client Interface

A job in workflow is represented by a set of interdependent tasks arranged in a Directed Acyclic Graph (DAG) [7]. 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.

Our main approach to autonomically service reconfiguration is performed in two steps. First, context-aware service discovery provides a set of services that are candidate to the configuration. Second, starting from the selected services and user task, context-aware process integration provides a set of configuration schemes that conform to the task's behavior further meeting all the context requirements.

5. Experiment Results

We present a simple example that describes how our autonomic service reconfiguration algorithm can be used in a UbiComp environment. This example 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 searching application using fault recovery strategy of FM. Our defined configuration property of searching application is shown in Fig. 5. Searching method-1 requires large memory but it takes short time to ends. Searching method-2 is slower than method-1 but it is a reliable task.

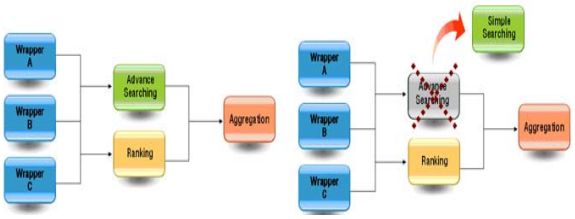
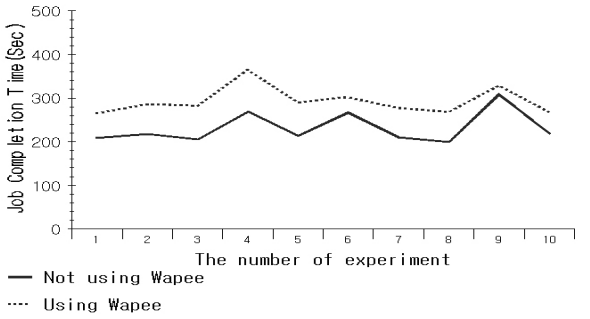


Figure 5. Our test scenario

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'.



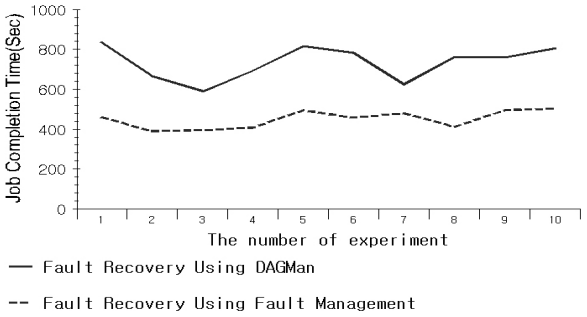


Figure 6. Performance Comparison using Wapee and non-Wapee of the Job

On Fig. 6 we showed the success rate and percentage of used fault-tolerance mechanism in Wapee. 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 below graph.

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 & Future Works

Wapee, an autonomic management middleware, executes likely faulty applications successfully with semantically meaningful strategies associating with service and fault ontology trees in ubiquitous 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 supports dynamic software component reconfiguration plan based fault and service ontology to provide fault-tolerance in a ubiquitous computing environment.

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. This allows better semantic interoperability between different context information on UbiComp environment. 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.

7. References

[1] M. Weiser, *The computer for the 21st Century* Scientific American, Vol. 265, No. 3, pp. 94-104, September, 1991.

[2] I. Foster. C. Kesselman, S. Tuecke. *The Anatomy of the Grid: Enabling Scalable Virtual Organizations* International J. Supercomputer Applications, 2001.

[3] Satish Tadepalli, Calvin Ribbens, Srinid Varadarahan *GEMS: A Job Management System for Fault Tolerant Grid Computing* High Performance Computing Symposium, 2004

[4] CORBA Fault http://www.omg.org/cgi-bin/apps/doc?formal/01-09-29.pdf

[5] 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

[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] Condor DAGMan http://www.cs.wisc.edu/condor/dagman/

[8] 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