

# Service-Oriented Approach for Distributed application Assembly and Management

E. Grishikashvili, N. Badr, and A. Taleb-Bendiab

School of Computing and Mathematical Sciences  
Liverpool John Moores University  
Byrom Street, Liverpool, L3 3AF, UK  
Email: {cmsegris; cmsnbadr; a.talebbendiab}@livjm.ac.uk

**Abstract** - Over the last few years, computer based communications have changed considerably due to major developments in the fields of networks, mobile devices, and services. Today's networks typically integrate fixed and wireless technologies to allow a broad variety of inter-networked services to be accessed by user in an uniform fashion. This paper presents on going work focused on the development of a framework for on-demand application service discovery, assembly and management.

## 1. INTRODUCTION

Over the last few years, computer based communications have changed a lot due to major developments in the fields of networks, mobile devices, and services. Today's networks typically integrate fixed and wireless technologies to allow a broad variety of inter-networked services to be accessed by user in an uniform fashion. In addition, the working behaviour of the users has changed. For instance, the increased user mobility requires mechanism to manage services and allow for their access from different places. Users want to be able to access the services from a wide variety of client devices, such as desktop systems, PDAs, mobile phones, and in-car computers. Furthermore these services should work together, interoperate.

These new user expectations are driving the researchers and developers to change the way they built application systems. Rather than creating large, monolithic applications or desktop-oriented, client/server applications, there is need to build applications using a service-oriented application design. Software is being broken down into its constituent parts – into smaller, more modular application components or services. These application services make use of infrastructure software that has also been decomposed into discrete system services. All these discrete services can be deployed across any number of physical machines that are interconnected. By reassembling a few services into a new configuration, a user can create a new service.

At the same time distributed applications are notoriously difficult to develop and manage due to their inherent dynamics, and heterogeneity of their implementation, topology, deployment and network requirements. Middleware technology has come to the rescue by easing and facilitating the development and interoperation of distributed applications. Until recently, however, little attention has focused on combining assembly, and control concepts in conjunction with middleware technologies to understand dynamic behaviour and assist with the runtime management of distributed applications.

The above described capabilities are directly obtainable by basing next generation of distributed systems on Jini [1]. The Jini architecture provides infrastructure elements required to federate and discover other services offering desirable functionality. Ultimately Jini gives us the capability to build, maintain, and alter a network of devices, software, and user.

However, building distributed systems that are highly adaptive, interactive, interoperable and autonomous requires more than just Jini technology. Putting together Service-oriented paradigm with its Architecture Description Language concepts, and XML technology are providing building blocks for that kind of systems. Although there are various works such as Rio, DASADA's "Service+Contract" [2,3,4] in this area and many are under development there are many problems yet to be solved.

This paper describe an approach that combines assembly and control services together to detect and correct runtime conflicts that may occur in end-to-end application services. The remainder of this paper is structured as follows: section 2 provides a brief introduction of Service-Oriented abstraction with the review of assembly and control services, which feature as main services in our framework. Section 3 provides description of the framework architecture. Section 4 describes a recent example application conducted to evaluate the framework. Finally,

section 5 draws overall conclusions and mentions future work.

## 2. SERVICE-ORIENTED ABSTRACTION

We regard the software components in a distributed application as self-contained binary implementations, consisting of one or more objects, which are hosted by a Jini-enabled device. Components communicate with each other through connectors that are implemented via software interfaces and through this view one may describe a distributed application using a *component-connector* abstraction [5]. However, individual components also use and provide services in order to deliver application service to the users. These services form the basis of an alternative *service-oriented* abstraction of a distributed application. In this abstraction an application is considered as a *federation* of services distributed over a network. A service represents a *logical* concept such as a printer, or chat service that can be discovered dynamically by clients and used according to a mutual contract of use. The service-oriented abstraction forms the bases of the developed framework, which provides a set of services that can be used to assemble, control and manage the application services of a distributed application.

### 2.1 Assembly Services

The concepts of *service assembly* can be simply explained as follows; when a user would like to implement a particular task, a situation might happen that the user's current computing device cannot satisfy all of the service requirements in order to solve particular task, e.g., laptop does not have installed application services such as calculator and currency inventor/ or text editor does not have the spell checker and the user needs these applications in order to write report about his business trip while he is in train coming back from this trip. One possible solution to this problem is attempting to use a collection of accessible computing resources around a user in order to realize a functionally equivalent application by combining all the required resources within a user's computer device.

The major advantages of this approach are: to increase the system flexibility and diversity by allowing multiple candidates for building blocks to construct an application and not only provide multiple options to create existing applications but also supporting the development of novel types of applications that consists of a variety of components [6].

Figure 1 shows our model of service assembly, which is explained further below.

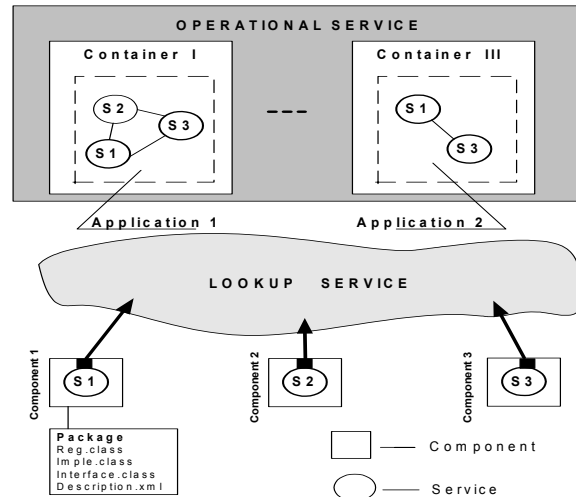


Figure 1: Service Assembly

The *Operational Service* is a Virtual Space containing one or more containers. A *Service Container* is a Virtual environment where services are executed and assembled ready for incorporation into a new application and each *Container* may contain one or more services. A *Component* is a set of all necessary class files necessary to allow service publication and service use via remote method invocation (components is typically stored in a jar file)

A *Service* is a logical concept that may be network and/or a software service. Service can be implemented and provided by a component for use by any other component. Services may be distributed across several different machines and accessed through Jini's discovery/lookup mechanism and used through Java's RMI protocol.

In Figure 3 *Component 1* offers *Service 1* by first registering it on a *Lookup Service* with a *Service Description* attribute, where the *Service Description* is an XML document and contains all information about the service. This information is needed by other components that may either implement the service or may need to invoke the methods of the service. On the first invocation of a service method, the service comes to life within the *Virtual Service Container*, which may contain a number of services. When the user needs two, three or more services to work together as an application, s/he makes a request for these services and the framework finds a suitable component capable of providing these services.

By using Jini's lookup service with a service name and attributes it is possible to find and parse the XML document describing the service (Service Description attribute). The XML document specifies the signatures of the methods implemented by the service. Any component that wants to use the service may parse the XML document component wanting to obtain the method signatures and invoke the service methods as required. The reconfiguration service is responsible for saving the information about different versions/ states of the services. Each different combination of services creates a new distributed application and the system creates and saves meta-information about each application. Service combinations are stored in XML format and the Service Manager oversees the deployment and subsequent operation of each new application through the use of monitor and control services, which together provide self-adaptive capabilities.

## 2.2 Control Services

Controller concepts and control services have recently gained popularity amongst the self-adaptive software

community, typified by [7], who use control services to adapt the structural configuration and dynamic behaviour of an application. Structural components can evaluate their behaviour and environment against their specified goals with capabilities to revise their structure and behaviour accordingly. Control services make use of well-specified control functions with *feed-forward* and *feedback* loops to enable a target application to be monitored to regulate its operation in accordance with its given control model.

Control services monitor behaviour, via information provided by assembly services. They then detect and identify conflicts and formulate remedial action in the form of a resolution strategy. Conflicts are identified and categorized according to type before a resolution strategy is used to minimize the conflict. A monitor element then provides feedback to guide the conflict resolution tasks, which are used to implement the conflict resolution process (Figure 2). The specific control service tasks and information exchanges are listed below.

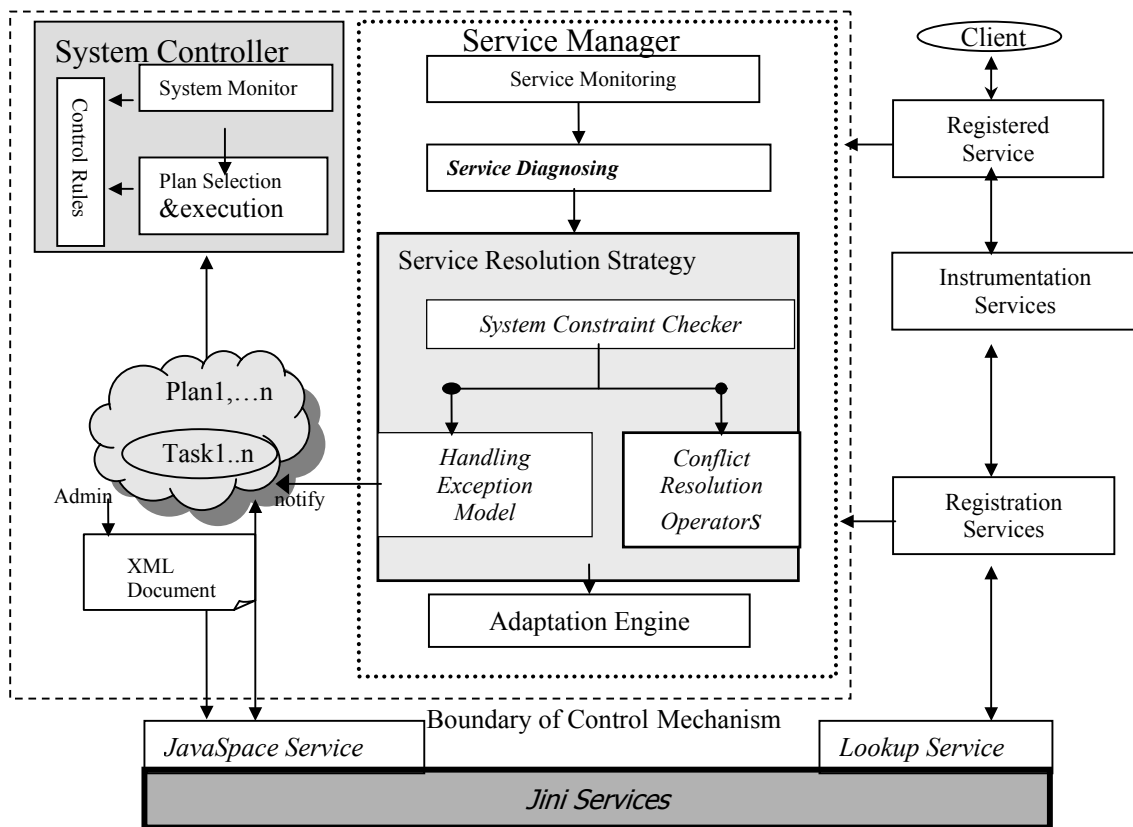


Figure 2: Conflict Resolution Process and Coordination

1. Monitor task: uses of a set of control rules against which behaviour is monitored to detect conflicts.
2. Diagnostic task: the execution of a control rule implies a conflict, which activates the diagnosis task that results in:
  - a. Identification of the part of the control rule that raised the conflict.
  - b. Identification of the cause of the conflict through the examination of messages intercepted by instrumentation services from which the attributes and methods of the offending object may be acquired using Java's reflection API .
  - c. Classification of the type of conflict, which provides the basis for the selection of a conflict resolution strategy.
3. Notification: uses Jini's remote event mechanism to notify clients and servers when conflict resolution solutions become available.
4. Control rules: serve as the basis for the previous monitoring and diagnostic tasks. They consist of a number of rules and gates that execute when a conflict is detected. This in turn may result in the firing of a remote event to notify of the availability of a resolution solution strategy.
5. Exception handling: uses Java's exception handling facilities to catch exceptions, which are thrown when a control rule executes due to a conflict that cannot be solved. Exceptions are dealt with according to priority, which may be low, intermediate or high to accommodate varying degrees of fault tolerance.

Conflict resolution strategies were specified using the *Design by Contract* approach of Meyer, [8], which is a discipline for defining precise checkable interface specifications for software components based on the theory of Abstract Data Types (ADTs).

### 3. FRAMEWORK ARCHITECTURE

The architecture of the framework shown in Figure 3, is based on a three –layer model. The first layer contains the core Jini middleware services, which include discovery, and lookup. The heart of the framework, contains the framework services of: Assembly Service, Control Management and Meta-information Services. Each of them are explained below:

- Registration Service: registers new network service on the local lookup service.
- Configuration Service: puts service on the container for the execution.

- Monitoring Service: monitors lookup service, operational service and new developed application
- Operational Service: maintains the support environment required by certain application services (e.g. Java Servlets require a web server is running). Contains different containers on different platforms. The container provides a simple environment to support services. Container can hold any number of services
- Assembly Services: is responsible for assembled services that are running on the container. It must register assembled services as a new application.
- Control service: examine information provided by monitoring service and described as a meta information about service. Control services use control rules, activated by conflict and select a conflict resolution solution strategy. Control services are considered further in [9]
- Meta-Information: maintains state information to provide a faithful “Up-to date” representation of an application at runtime, which can be accessed by any of the above services, especially by Control mechanism.

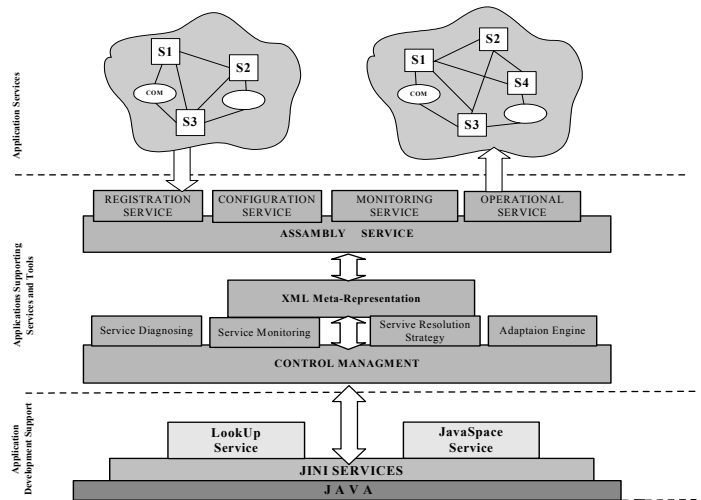


Figure 3: Framework Architecture

Finally, the third layer contains the application services, which are assembled configured, monitored and controlled, at runtime, by second layer. As the new developed application is a combination of different services in case of failing one of them the system is able to adopt changes and reconfigure the application runtime.

#### 4. EXAMPLE APPLICATION

In order to demonstrate the idea, an example of home appliances in wide and home network is used. The user has home appliances connected to web. Before she leaves work, remotely from her office she decides to assemble an impromptu application in order to make her appliances to work together. (Fig. 4) Finds the services the appliances are offering, using the registration service registers them in order to make them interactive. She knows she needs to wake up 8:00 am and have breakfast before leaving home.

So she needs clock, lamp, toaster and kettle to work together. The new application is monitored and managed by system.

As there is dependence between these services if one of them fails then whole application goes wrong. In this case the system takes care of this federation of services and without user intervention (user is sleeping) finds another clock on the lookup service and replace failed one. As a result the system makes changes and reconfigures itself.

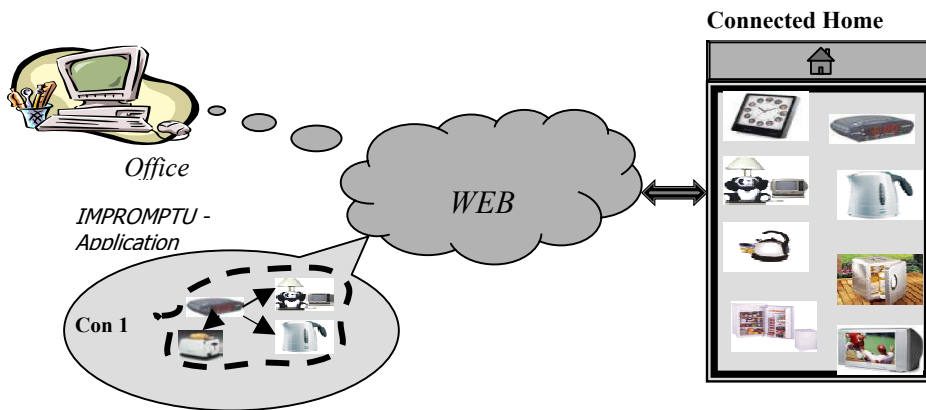


Figure 4: Home appliances scenario

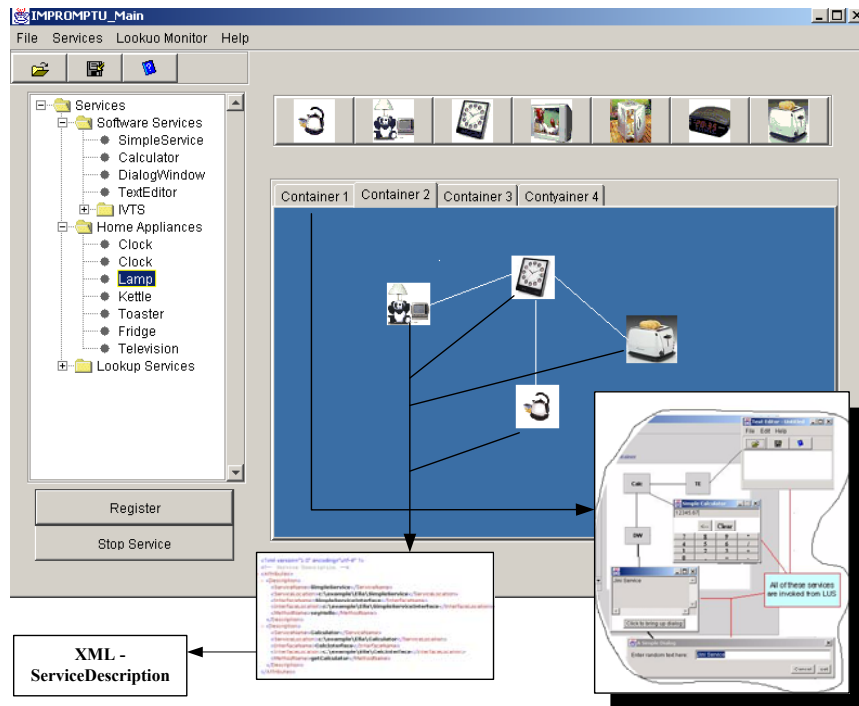


Figure 5: Prototype

As illustrated in Figure 5 the GUI the user can create different applications in different containers. The application – “Home Appliances ” was created in container 2, in Container 1 the user can make another aggregation of services combining a Text editor and a Spelling Checker, or a Calculator and Currency converter. Each service has XML format description file containing description of invocation method. After parsing this file the system is able to invoke the service. In the case of software services that contain different methods for invocation, the description file provides location of actual class file and using Java reflection the system is able to find the methods for invocation.

A new XML description file is generated, according to the user’s requirements that shows which container contains what services and describes the dependencies between them. The XML description file is passed to the manager, after which, the control mechanism takes care of the system.

The System Manager (Fig.2) is responsible for monitoring the service behaviour and if conflict arises the manager tries to diagnose, identify the type of conflict. For instance, when the manager is monitoring the home appliances application and diagnoses the failure of clock service that does not send any message to the other appliances to start on certain time, the manager check service constraints and decides to replace that service with alternative one. At this point the role of a manager service leads to storing notification about the service’ data structure on the shared space. During that time the system controller keeps monitoring the whole system and checks regularly the shared space looking for notification sent by service manager. The controller selects the suitable system plan for instance, requiring the other home appliance will connect with new clock instead of the failed one and the system work again without the user.

## 5. CONCLUSIONS

In this paper, we have described a framework, based on the combination of assembly and control services. We have described a Jini-based architecture, based on a three-layer model through which control services may adapt behaviour and performance by activating conflict resolution strategies based on information provided by assembly services. We

have demonstrated the application of the framework through a simplified example of networked appliances. The prototype has been implemented and tested not only for network appliances, but also it considers the software services, as they are more difficult to manage (Fig5).

We intend to continue the development of service-oriented framework and its evaluation and extend its capabilities to provide security services.

## REFERENCES

- [1] Jini Community, <<http://www.jini.org>> (accessed January 2002).
- [2] Jini Community, “Rio Architecture Overview”, Rio Project, <<http://www.jini.org/projects/rio>> (accessed January 2002).
- [3] Wells, D.L. (Object Services and Consulting Inc.) and Nagy, J. (Air Force Research Laboratory), “Gauges to Dynamically Deduce Componentware Configurations”, DASADA Project List, DARPA (Program Sponsor), <<http://schafercorp-ballston.com/dasada/projectlist.html>> (accessed January 2002).
- [4] Nathan Combs, BBN Technologies “Reliable Recruitment and Assembly of Peer-to-Peer Services and Distributed Workflow”, 2000
- [5] Bieber G, Carpenter J, ”Introduction to Service-Oriented Programming”. Motorola ISD, 2002
- [6] E.Grishikashvili, N.Badr, T.Bendiab “From Component Based to Service Based Distributed Applications Development and Lif-time Management”. EUROMICRO’03, September 2003
- [7] Osterweil, L.J. and L.A. Clarke, Continuous Self-Evaluation for the Self-Improvement of Software, in 1st International Workshop on Self-Adaptive Software (IWSAS2000), Oxford, UK, Springer-Verlag,2000
- [8] Meyer, B., Object-Oriented Software Construction, 2nd ed, Prentice Hall, New Jersey, 1997
- [9] Badr, N., D. Reilly, and A. Taleb-Bendiab, A Conflict Resolution Control Architecture for Self-Adaptive Software, in Proceedings of the International Workshop on Architecting Dependable Systems: WADS 2002 (ICSE 2002), Florida, USA, 2002.