A Novel Service-Based Paradigm for Dynamic Component Integration

Sudhakiran V. Mudiam and **Gerald C. Gannod**[*] [†]
Dept. of Computer Science & Engineering
Arizona State University
Box 875406
Tempe, AZ 85287-5406
E-mail: {kiranmvs,gannod}@asu.edu

Timothy E. Lindquist
Dept. of Elect & Comp Engg Technology
Arizona State University - East
7001 E. Williams Field Road, Building 50
Mesa, AZ 85212
E-mail: tim@asu.edu

Abstract

We are developing a novel service-based paradigm for dynamic component integration that facilitates the creation of *Intelligent Services* from COTS (Commercial Off The Shelf) components, legacy components, and application frameworks by using techniques such as mediation and adaption or "wrapping". This framework supports the construction of applications by dynamically integrating these services at run-time based on available resources and allows for a federation of services that can evolve over time. As a part of the ongoing research effort we are utilizing an architecture based specification language that enables us to automate the process of creating these intelligent services.

Introduction

In the past, software reuse, and especially Component Based Software Engineering (CBSE) have been in the spotlight as various component technologies have matured. CBSE is much more than simply using object request brokers, setting up a library of useful code, or modular development. It involves building, acquiring, assembling and evolving systems. Traditionally, in CBSE technologies, several components are packaged together to create software systems. Emerging concerns include multiple suppliers of components that provide the same functionality, coping with multiple versions, and configuration of components. Currently, CBSE addresses issues and technologies such as Commercial-Off-The-Shelf (COTS) components, in-built components, and application frameworks. Emerging technologies of component integration include, Enterprise Java Beans (Thomas 1998), CORBA (Obj 1996), Jini (Richards 1999), Microsoft's DNA(Kirtland 1999), DCOM(Eddon & Eddon 1998) and IBM's San Francisco (Monday, Carey, & Dangler 1999). All these technologies provide a component model where a predefined infrastructure acts as "plumbing" that facilitates communication between components. Tools and environments supporting these technologies are being widely used in the industry and continue to provide many benefits.

We are developing a novel service-based paradigm for dynamic component integration that facilitates the creation of *Intelligent Services* from COTS (Commercial Off The Shelf) components, legacy components, and application frameworks by using techniques such as mediation and adaption or "wrapping". This framework supports the construction of applications by dynamically integrating these services at run-time based on available resources and allows for a federation of services that can evolve over time. As a part of the ongoing research effort we are utilizing an architecture based specification language that enables us to automate the process of creating these intelligent services (Gannod, Mudiam, & Lindquist 2000).

The remainder of this paper is organized as follows. The next section introduces our approach for application construction based on the use of intelligent services. An example using the approach is described in the following section, and the balance of the paper discusses related work, conclusions and future investigations.

A New Paradigm

We are developing an approach based on a new paradigm that looks at software reuse from a different perspective in which components are viewed as services available on a network. In this paradigm, components are dynamically composed into federations to make up an application (Mudiam 2000). The key features of our approach within this paradigm are:

- Components of varying granularity are bundled as "intelligent services" and made available on a network.

- The paradigm provides an integration framework, or middleware, that allows for the dynamic integration of these components (bundled as intelligent services) at run-time.

- The paradigm facilitates the use of various patterns of interaction (architectural styles) between clients and intelligent services.

- The intelligent services provide a clear set of interfaces that are discovered dynamically at run-time and achieved using filters and adapters.

The process of component integration consists of the following steps.

1. Specification of the components as services.

2. Generation of the services along with the appropriate adapters and storing them to a repository.

3. Specification of a client to make use of services from the repository.

4. Generation of the client.

5. Execution of the client, performs the integration of the specified services at runtime.

Steps 1 and 2 are typically performed only once for each service while Steps 3 through 5 are performed as needed for each application. One of the primary goals of this work is the use of automatic synthesis to generate the source code necessary to achieve service integration. Our preliminary investigations have yielded an approach for generating wrappers of legacy applications for use as services (Gannod, Mudiam, & Lindquist 2000). In this work, a specification of a legacy software is created that defines the interface to a potential service. Second, the appropriate adapter source code is then synthesized based on the specification.

In order to specify how applications are to be integrated, we use the ACME architecture description language (ADL) (Garlan, Monroe, & Wile 1997) to describe both the available services as well as the client components that utilize those services. These specifications capture the interface of the components in such a way that the services can be regarded as black-boxes while remaining loosely coupled from implementation details.

Once the services are generated and stored in a repository, clients can make use of these services by identifying a general class of service that is required and the architectural style is necessary to properly interact with a client. This work addresses several issues that need to be considered during application specification and construction including:

- The specification of application architectures

- The specification of a *component type* in order to manage style interactions

- Graphical user interface integration including the use of shared graphical features

Again, using the ACME ADL, an application architecture is specified in order to describe the client composition and to identify the service classes that are needed by the application. Using this specification, the source code necessary to realize integration is generated (but is not bound to specific services). The remaining source code for the client, as with all applications, is provided by the user.

Integration of all services in our paradigm is performed with the execution of clients as each service becomes available. At first a client registers with a lookup service. Once services become available and join the network, the client is notified. The client integrates with the services by performing the GUI component integration as well as the service adapter integration and utilizes the service.

The research described above makes use of Jini (Richards 1999) technology to realize service integration. The components are wrapped as Jini services and we make use of the discovery and join mechanism to enable services join a Jini network.

Example

Figures 1 and 2 depict specifications of a client component and a number of services, respectively. In this example, the client consists of an editor that needs to utilize version management, printing, and compiling services. When the client runs, it joins the Jini network using Jini's discover and join protocol. As shown in the specification in Figure 1, the services that are required by the client are described as ports to which services will be bound.

```
Component Editor = {
  Properties {
    Part-of-client :
      string = "true";
    GUI-CodeFile :
      string = "ClientGUICode.java";
    Component-type :
      string = "Call Return";
  };
  Port VM_Port = {
    Properties {
      Port-type :
        string = "caller"; };
  };
  Port COMPILING_PORT = {
    Properties {
      Port-type :
        string = "caller"; };
  };
  Port PRINTING_PORT = {
    Properties {
      Port-type :
        string = "caller"; };
  }; };
```

Figure 1: Client Specification

The specifications of the services, as shown in Figure 2, describe each of the available services in the integration network. In this example we show only the specification of the RCS service and provide stubs for other services. Note that the other services would indeed be specified in the same manner as the RCS service. Each of the port specifications in a service com-

ponent describe associated services. As such, the RCS service could potentially have sub-services for check in and check out.

```
Component RCS = {
Properties {
   Component-type :
   string = "Call Return";
};
Port ChkIn = {
  Properties {
   Signature :
      string = "String filename,
       String params, String filedata ";
   Return :
      string = "Boolean result";
   Cmd :
      string = "ci + filename + params";
   Pre :
      string =
      "WriteFileData(filedata,filename)";
   Post :
      string = "";
   Interface :
      string = "VersionManagement";
   Path :
   string = "C:\\RKTOOLS\\BIN\\CI.EXE ";
   Port-type :
      string = "callee";
}; }; };
Component Lzpr = {
  Properties {
   Component-type :
      string = "Call Return";
  };
  Port Print = {
    Properties {
    ......
}; }; };
Component Javac = {
  Properties {
   Component-type :
      string = "Call Return";
  };
  Port Compile = {
    Properties {
    ......
}; }; };
```

Figure 2: Services Specification

Initially, when no services are available, the client only has its editing GUI component as shown in Figure 3. However, since the client has access to the list of services that it needs, it utilizes the lookup services on the Jini network in order to complete service integration. As each service comes up and joins the Jini network, the client learns its existence and integrates them into the client. The Figure 4 shows the result of the integration once all the services have joined the network. In this example, we have a version management service called the *RCSService*, a compiling service called *JavacService* and a printing service called *lzprService*.
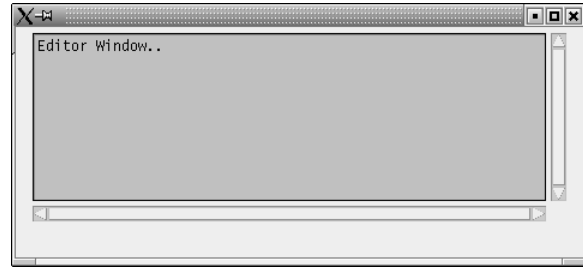

Figure 3: Initial Client

Related Work

Sullivan et al. look at systematic reuse of large-scale software components via static component integration (Sullivan *et al.* 1997). That is, they use an OLE-based approach for component integration. To demonstrate the use of their scheme, they developed a safety analysis tool that integrates application components such as Visio and Word. In our approach we use a dynamic approach for component integration and thus, can utilize a wide variety of components whose interfaces are discovered at run-time.

CyberDesk (Dey *et al.* 1997) is a component-based framework written in Java that supports automatic integration of desktop and network services. This framework is flexible and can be easily customized and extended. The components in this framework treat all data uniformly regardless of whether the data came from a locally running service or the World Wide Web. The goal of this framework is to provide ubiquitous access to services. This approach is similar to our proposed approach in that they use a dynamic mapping facility to support run-time discovery of interfaces.

Conclusions

The availability of components on a network that provide services at run-time has many potential applications including the use of heterogeneous components executing in distributed environments. Enabling technologies such as Jini allow for dynamic component integration. We have described an architecture-based methodology of building components for reuse, and later using them with in a larger context to build applications.

We are currently building an environment for supporting the specification and synthesis of applications that utilize services through dynamic integration. The current state of this work includes tools for sythesizing both service and client integration code. We have used these tools to generate examples including the one described in this paper. Our future investigations include developing an approach for dynamically generating mediators to facilitate service interactions that bypass the need to communicate via client applications.

References

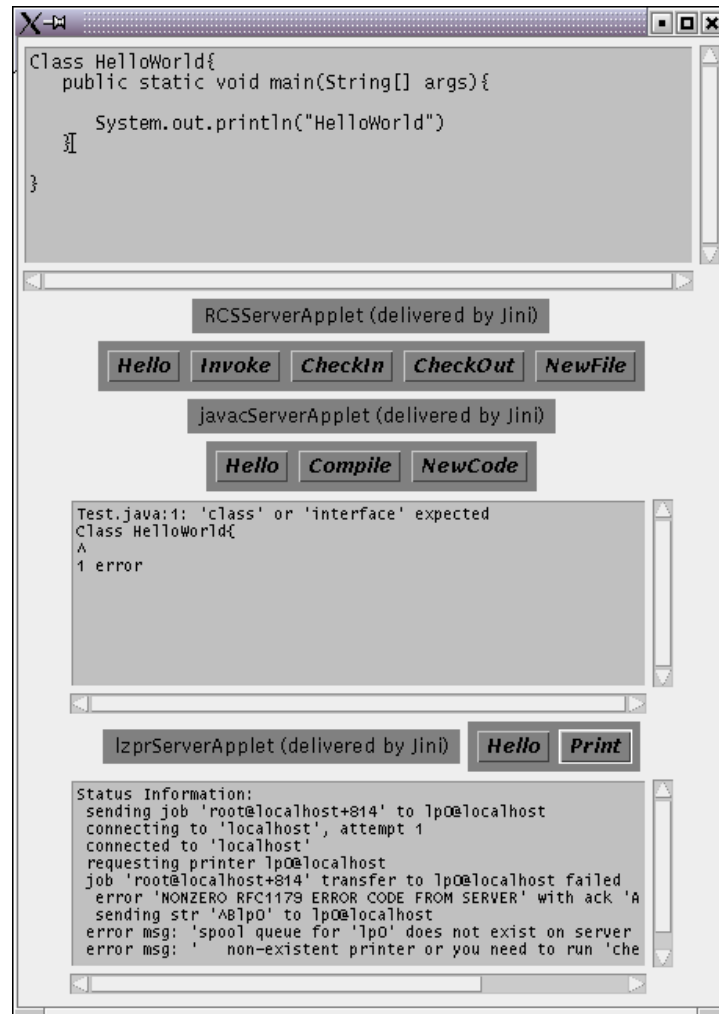Dey, A. K.; Abowd, G.; Pinkerton, M.; and Wood, A. 1997. Cyberdesk: A framework for providing

Figure 4: Client After Integration

self-integrating ubiquitous software services. Technical Report GIT-GVU-97-10, Georgia Tech.

Eddon, G., and Eddon, H. 1998. *Inside Distributed COM*. Microsoft Press.

Gannod, G. C.; Mudiam, S. V.; and Lindquist, T. E. 2000. An architecture-based approach for synthesizing and integrating adapters for legacy software. In *Seventh Working Conference in Reverse Engineering*, 128–137. IEEE Computer Society.

Garlan, D.; Monroe, R. T.; and Wile, D. 1997. Acme: An Architecture Description Interchange Language. In *Proceedings of CASCON'97*, 169–183.

Kirtland, M. 1999. *Designing Component-Based Applications: Build Enterprise Solutions with Microsoft Windows DNA*. Microsoft Press.

Monday, P.; Carey, J.; and Dangler, M. 1999. *San-Francisco Component Framework: An Introduction*. Addison-Wesley.

Mudiam, S. V. 2000. A novel service based paradigm for dynamic component integration. Ph.D. Proposal, Arizona State University.

Object Manangement Group. 1996. *CORBA: Architecture and Specification V2.0*, formal/97-02-25 edition.

Richards, W. K. 1999. *Core Jini*. Prentice-Hall.

Sullivan, K. J.; Cockrell, J.; Zhang, S.; and Coppit, D. 1997. Package oriented programming of engineering tools. In *Proceedings of the International Conference on Software Engineering*, 616–617.

Thomas, A. 1998. Enterprise java beans technology. Technical report, Patricia Seybold Group.