Dynamic Architectural Changes for Distributed Services

Paolo Falcarin, Patricia Lago, Maurizio Morisio
Dipartimento di Automatica e Informatica
Politecnico di Torino, Italy
c.so Duca degli Abruzzi 24, I-10129
{Paolo.Falcarin, Patricia.Lago, Maurizio.Morisio}@polito.it

Abstract

The design of complex software architectures for distributed systems always faced different problems in both development and maintenance. Design decisions like the kind of architectural style, the middleware to use, and the interaction styles among distributed components are variants often chosen in the early design phases. Hence, when some changes are needed, rollback is difficult and expensive. Moreover, when a developer team implements the system, it is difficult to maintain coherence in source code with the architectural specification; this implies a longer phase of debugging and re-designing. A different strategy can be based on delaying as much as possible these design decisions, to be able to choose the best architecture and middleware during prototyping. This approach permits a more agile development process that allows choosing among possible alternatives before deployment, or even after deployment, and changing these variants at runtime. This paper describes JADDA (Java Adaptive component for Dynamic Distributed Architectures), a software component we developed to cope with these issues with a minimal impact for developers.

1. Introduction

Recent trends in software development show an increasing request for variability, i.e. software locations where behavior and structure can be configured. Handling variability implies the need of delaying design decisions in order to decrease the number of decisions that could be irreversible or very expensive to rollback. Software variability is the ability of a software system to be changed, customized or configured in order to be easily adapted to different contexts. This implies that software is more reusable, that it is designed to support evolution, and that it can be easily used in an agile software development process [5].

Starting from these open problems, in this paper we focus on distributed software architectures, and try to give an answer to these related questions:

- How can we delay design decisions like the architectural style, the middleware, and the interaction style (e.g. synchronous or asynchronous) of components?
- How can we write source code that is independent from the kind of middleware used?
- How can we check if our source code adheres to the architectural specification?
- How can we apply fast design changes, when development shows design errors?
- How can we check if a certain architectural constraint holds in the implementation?

- How can we evaluate alternative implementations, in early prototyping?

To find an answer to the questions above, we developed JADDA [2], a software component supporting dynamic management of variability.

2. JADDA

JADDA is a software component that relies on an architectural specification defined in xADL (XML-based Architecture Description Language, [1]). Among the various ADLs focused on dynamic software architectures, we chose xADL because of its characteristics:

- It is designed to be a standard way to express architectural specifications.
- It is extensible and adaptable to the kind of ADL one is used to work with: it allows architects to apply extensions by defining new XML-Schemas that can be referenced in a xADL file.
- It is based on XML, which makes it easier to parse by JADDA source code than other ADLs.

We have reused the xADL basic schema to define the architecture topology and we have extended xADL with new XML-schemas to specify information specific to distributed systems. Some of these schemas are used to provide information about different middleware protocols, while others define information typical for distributed systems. Another important characteristic of xADL is the ability to generate Java code from different schemas; This is possible by means of the APIGEN tool available at the xADL website [4]. Generated source code can be used by applications (in our case by JADDA) to handle XML-schemas data inserted in a xADL file.

Our XML-schemas are based on the definition of Component, Connector, and Link as given by xADL:

- A Component is an object that makes some computation and offers a set of interfaces.
- A Connector is an object that focuses on data communication among components.
- Links are semantic-free connections between interfaces on components and connectors that indicate the topology of architecture: each link represents a directed connection showing that a component needs to call a method on an interface of another component.

Starting from these definitions, we have extended the characteristics of Connectors and Links(defined by xADL in its own XML-schemas). We specified these extensions in two corresponding new schemas: Distributed-Connector and Distributed-Link: these contain additional data defining remote interactions among distributed components. *Distributed-Connector* is the basic schema that is specialized by other XML-schemas related to middleware protocol standards, like CORBA-Connector for CORBA-IIOP [6], and SOAP-Connector for SOAP [7]. For example, the CORBA schema defines tags like:

- NameServer: it includes all the information needed for binding and retrieving CORBA references with a certain implementation of the CORBA Naming Service;
- Location: specifies the runtime information to connect to the Naming Service (e.g. hostname and port);

- Stub, skeleton, objectAdapter: contain prefixes and suffixes added by a certain CORBA implementation to the client stubs, server skeletons and object adapter, for JADDA to make remote invocations with Java reflection.

A standard protocol like CORBA-IIOP can be implemented by different middleware platforms, offering slightly different APIs to applications. Therefore, including in the same architecture different kinds of CORBA implementations, means having different instances of CORBA-connectors in a xADL file: each CORBA connector defines tag values (in the CORBA-connector schema) to qualify its own specializations.
The kind of middleware used by a component interface is defined in the *extended-Interface* XML-schema. This schema extends the xADL's Interface schema. It contains the reference to the connector instance used by a component interface and other information needed to check architectural constraints. For example, it is possible to define the sequence of method invocations allowed on an interface; this specification can be used by JADDA to check if an application is using a remote interface in the right way.
A *Distributed-Link* schema represents a direct relationship between a component and the interface of another component. This schema also specifies the interaction style (synchronous, or asynchronous) used by this link, and the list of interface methods that are invoked with a different style. This approach also allows in the future defining new interaction styles. Before making a remote invocation, JADDA checks in the xADL file the links that exist and the style to use: if the searched link is not defined, it means that the remote invocation is not allowed by the architectural specification, and hence it does not occur. This event is reported to the developer and to the designer, who can react in two different ways: either the system architect changes the specification, or the developer changes the implementation.
In order to support dynamic architectural changes, each application must include an instance of JADDA and use it to make remote invocations using the JADDA API.
During initialization JADDA registers in the JADDA System administrator console in order to receive through the network the current xADL architectural specification file. Next, the remote interface references are stored by the CORBA Name server or by the UDDI registry (depending on the information contained in the xADL file).
Application independence from the middleware in use is achieved in JADDA by wrapping the different middleware protocols for remote method invocation, like CORBA-IIOP, and SOAP. An example of typical usage looks as follows:

```
Jadda  jadda = new Jadda();
String componentName = "ChatServer";
String interfaceName = "ChatManager";
String methodName = "accessRoom";
String parameter = "Joe";
jadda.call(componentName, interfaceName, methodName, parameter);
```

The method "`call`" is overloaded in order to offer different versions able to call methods with different parameters; in the different "`call`" method signature, the first three strings identify the requested method, and the fourth parameter carries Java "Object" types: they all refer to the main "`call`" method implementation with the following signature:

```
Object call(String component, String interface, String method,
Object[] param);
```

The method "`call`" searches in the xADL file the information about the middleware needed to communicate with the requested interface method, and it uses Java reflection to make the remote method invocation.
Using this approach, we provide a more abstract view of different interactions, allowing programmers to use different libraries implementing connectors, i.e. different APIs to make remote method invocations, depending on the middleware implementation. This implementation strategy reduces significant problems in the development and maintenance of software systems, and the connector source code is well separated from the component source code. In this way, the source code of software services is more portable and independent from the underlying middleware. In addition, the service source code can be easily reused or upgraded. Abstracting remote invocations with local methods is an approach used in the container architecture of JBoss [8] application server: the developer has to implement the "`ContainerRemote`" interface to implement adaptations for the desired middleware, in addition to the default implementation of RMI. Instead in JADDA these adaptations for CORBA and SOAP are already implemented and architectural specification is also checked out before every remote invocation.

3. Dynamic architectural changes

An XML-based specification can be easily parsed from source code. This means that an application can check at runtime if its own behavior is compliant with the current architectural specification. As we do not want to assign this task to the developer, this work can be delegated to JADDA, which monitors every remote invocation, checking if the architectural specification is maintained and if constraints hold.
In order to support dynamic architectural changes, JADDA implements a separate thread listening to the network for a new version of the architectural specification: in this way, the system designer using the JADDA System Administrator console, can change the xADL file and send it on the network to the involved services, which are immediately adapted to the new architecture. This allows trying different possible values for variants like architectural styles, middleware and interaction styles in a later phase of design, just before testing and deployment. Changes to these variants can be also applied at runtime, without rewriting or adapting any code in the different services.
Changing the xADL specification file during prototyping reduces the amount of development effort and returns quick results early in the process: try alternative implementations, it is possible to decide which architectural styles, standards, interfaces, middleware or even components to use in the final deployment. This approach makes development more agile, as promoted by extreme Programming. If later in the process, a chosen architecture or interface turns out to be inadequate, with JADDA, changes in the requirements lead to shorter delay in the development than with traditional software development. Moreover, during development, the adding or removal of constraints set on the methods of an interface can be used to help the developers in maintaining source code coherent to the specification.
On the other hand, changing the specification at runtime allows:

- Substituting a new version of the middleware
- Substituting the middleware used in a link, whenever the server-side component of the link changed the middleware used to publish its own interfaces;
- Notifying all the components dependent on a particular component X, that a new version of X has been deployed: this allows using hot-swappable components without changing other dependent components.

4. Conclusions

This paper describes JADDA, a component used to check architectural specification of distributed systems at runtime. JADDA reduces the effort and time needed in developing services based on different middleware: applications can use the same abstract API hiding the details of different middleware implementations, and a different kind of middleware can be used on each link to a remote interface. This cost reduction is obtained thanks to the three main features of JADDA:

1. An XML architectural specification based on xADL, and extensible with new XML-schemas that add middleware information;
2. Middleware abstraction in the application source code, bringing service portability and the ability to change middleware and interaction styles without modifying the application code;
3. Support for dynamic change of the architectural specification to adapt the application at runtime to new architectural requirements.

JADDA adaptability can be improved by allowing the dynamic downloading of new middleware components when a new version is available. Current work focuses on the extension of JADDA to rely on PROSE [3], a dynamic aspect-oriented platform able to insert and withdraw aspects at runtime. With this extension we expect to totally decouple the application code from middleware and architectural concerns.

Acknowledgements

The authors want to thank Eric M. Dashofy (from the Institute for Software Research of University of California, Irvine) and Isabella Vespa (from the Politecnico di Torino, Italy) for the help given with xADL.

References

[1] E.M. Dashofy, A. van der Hoek, and R.N. Taylor. *An Infrastructure for the Rapid Development of XML-based Architecture Description Languages*. In Proceedings of the 24th International Conference on Software Engineering (ICSE2002), Orlando, Florida.
[2] JADDA website. URL: http://softeng.polito.it/jadda/
[3] PROSE (PROgrammable Service Extension). URL: http://prose.ethz.ch/
[4] xADL 2.0 website. URL: http://www.isr.uci.edu/projects/xarchuci/
[5] Agile development website. URL: http://www.agilealliance.org/
[6] CORBA standard. URL: http://www.CORBA.org
[7] SOAP protocol. URL: http://www.w3.org/TR/soap12-part1/
[8] JBoss website. URL: http://www.jboss.org/