

The Bio-Networking Architecture Bi-weekly report #9 (Sep. 30, 2002): A Component Service Description Framework (CSDF) for Dynamic Service Composition

PI: Tatsuya Suda (suda@ics.uci.edu)
University of California, Irvine, <http://netresearch.ics.uci.edu/bionet/>

Introduction

In the Bio-Networking Architecture, complex applications emerge through autonomous interactions of multiple service components (called “cyber-entities”), similarly to individuals cooperating to achieve a complicated task in the real biological world. This concept of composing an application through an autonomous interaction of multiple service components at runtime is called Dynamic Service Composition.

Dynamic Service Composition reduces application development time as a large-scale complex application is autonomously created from simpler service components. Applications created through Dynamic Service Composition are flexible and adaptable. For example, depending on a specific type of a display device available to a user (e.g. a small PDA display or a full color PC display), Dynamic Service Composition allows creation of applications using a service component that matches the capability of the display device (e.g., simple interface for a PDA display, full color and high resolution interface for a PC display). In addition, in dynamically creating applications, service components may be selected based on various resource constraints and system configurations (e.g., CPU speed, memory size, network bandwidth) so that applications functionality and performance adapt to the context (i.e., various resource constraints and system configurations).

In order to create an application from several service components, a service component must be able to understand the functionalities and semantics of other service components to evaluate whether it can interact with them to compose an application. However, since service components are typically executable binaries, it is often difficult to obtain the functionalities and semantics of the service components from their codes. In order to solve this problem, the PI proposed that a service component carries a meta object describing its functionality and semantics and also designed a description language called “Component Service Description Framework (CSDF)” through which a service component designer can describe service component’s functionality and semantics in the meta object. With the use of CSDF and the meta object in service components, the functionalities and semantics of service components may be obtained at runtime, and thus, are used to dynamically compose an application.

New Achievements

The preliminary design of Component Service Description Framework (CSDF) has been completed, and the compiler for CSDF (to generate necessary Java source code from a given CSDF file) has been implemented. The specification of CSDF and the beta version of the CSDF compiler are available on the following web.

- <http://netresearch.ics.uci.edu/bionet/csdf/>

Component Service Description Framework (CSDF)

Component Service Description Framework (CSDF) is a framework designed for describing service components used in Dynamic Service Composition. CSDF describes Methods, Variables and Assertions of a service component.

A Method defines the service that a service component provides. A single service component may

have multiple Methods as it provides multiple services (e.g. a bank service component providing “deposit” and “withdraw” services). A Method consists of method name(s), input arguments, and output values. Both input arguments and output values are defined as a set of elements, namely, name, type, and attribute. In CSDF, a set of specifications called Flexible Interface Definition is used to declare a Method of a service component in a flexible manner so that service components with slightly different Methods can communicate and interact with each other.

CSDF also supports declaration of Variables of a service component. A Variable is defined as a set of variable name(s), type (such as integer, string etc), value and attribute. CSDF does not specify how a Variable can or should be used. A Variable may be used to indicate the state of the service component (e.g. “this speaker component is now being used.”), the configuration of the service that the service component provides (“the stereo speaker volume is set at level 5.”), and the description of the service component (“this is manufactured by Sony.”).

An Assertion is a Boolean condition associated with either a Method or a Variable of a service component. A Method may have two types of Assertions: Precondition, a condition that must be satisfied BEFORE the Method is invoked (e.g. “the Variable A must be 1 before this Method is invoked”), and Postcondition, a condition that must be satisfied AFTER the Method is invoked (e.g. “the Variable B must be positive after this Method is invoked”). A Variable may have an Assertion called Invariant, a condition that the Variable must always satisfy (e.g. “the Variable C must be greater than 5”).

There are several CSDF features that make CSDF unique and suitable for Dynamic Service Composition. First, CSDF supports Flexible Interface Definition to ensure that a service component can communicate and interact with other service components (that may be independently developed) to collectively provide an application. Details of Flexible Interface Definition are described in the bi-weekly report #4 submitted on July 22, 2002. Second, CSDF supports Assertion of Methods and Variables, such as Precondition, Postcondition, and Invariant. Assertion is useful and sometime necessary for Dynamic Service Composition. For instance, Preconditions and Postconditions may be used in order to guarantee that Methods may be invoked in a logical order. By properly setting Pre/Postconditions, one can detect that a certain Method (e.g. “read emails”) cannot be executed before a certain Method (“login”). Preconditions and Postconditions can also detect and prevent illegal Method invocation (for instance, to deposit a negative amount of money) in advance. Lastly, CSDF is language independent. In other words, CSDF may be incorporated in any language. This increases flexibility in both designing and implementing service components. The PI specified CSDF in an XML Schema style and also in a Javadoc style and has developed a compiler that parses a CSDF file in both styles.

Component Service Description Framework (CSDF) Compiler

The PI has developed a compiler for CSDF. The CSDF compiler parses a CSDF file and generates several Java classes, including an abstract class, a Proxy class, and helper classes. The CSDF compiler also generates a meta object from a CSDF file. A meta object is a Java binary object that represents Methods, Variables and Assertions declared in the CSDF file.

The CSDF compiler and the generated classes are used in the following manner to implement a service component. When a programmer implements a service component, he/she first designs Methods, Variables and Assertions of the service component and describes them into a CSDF file. Next, he/she compiles the CSDF file to generate Java classes and a meta object. Then, he implements the service of the component by extending the generated abstract class. Lastly, he/she packages all the class files along with the meta object into a single file, which is deployable to the Bio-Networking Architecture.

Current Status and Future Work

The PI is currently developing a mechanism for composing an application from multiple service components carrying CSDF data. The PI is also developing a simulator to evaluate the performance of the composition mechanism being investigated.