Towards a Programming Language for Services Computing

Arun Kumar
IBM India Research Laboratory
4, Block C, Institutional Area,
Vasant Kunj, New Delhi 110070, India
kkarun@in.ibm.com

D Janakiram
Dept. of Comp. Sc. & Engg.
Indian Institute of Technology Madras,
Chennai-600036, Tamil Nadu, India
d.janakiram@cs.iitm.ernet.in

ABSTRACT

Services Computing is emerging as a new discipline. The acceptance of web services technology stems from the fact that services enable easy integration and interoperation of enterprise level distributed systems. However, currently software developers are forced to translate business level service requirements and encode them into programs using low level abstractions such as objects. We propose to introduce language constructs for Service Oriented Programming that would enable raising programming abstractions from objects to services.

Categories and Subject Descriptors: D.3.3 [Programming Languages]: Language Constructs and Features [data types and structures, inheritance]

General Terms: Languages, Design

1. INTRODUCTION

Interoperability and manageability in the presence of heterogeneity are two very important concerns for enterprise IT system managers. Services computing promises to ease these problems by creating an infrastructure of loosely coupled components residing in an heterogeneous framework. This has lead to several research efforts that explore ways of increasing the level of automation in service discovery, invocation, composition and interoperation [1, 4].

However, the current services based software development model suffers from several drawbacks. The developers are forced to take into consideration dynamics of the runtime environment since services are actively running components rather than passive function/class libraries. Further, the developers are required to map high level service requirements to programming constructs available in current OO languages such as Java and C#. There is a clear mismatch between the needs of services software developer and the programming models available today. In this paper, we propose new language constructs that enable programming with services and discuss some benefits arising from such an approach.

2. PROGRAMMING WITH SERVICES

In this section, we build upon our previous work [3] that proposed granting a first class status to services. The proposal used a semantic web services based representation model [2] that applies the abstraction principle of classification to services and defines the concepts of service types and service instances [2, 1]. Service types capture the semantic description of the service primarily in terms of its interface description. A service instance, on the other hand, provides an operational description of a service and represents an actual running instance.

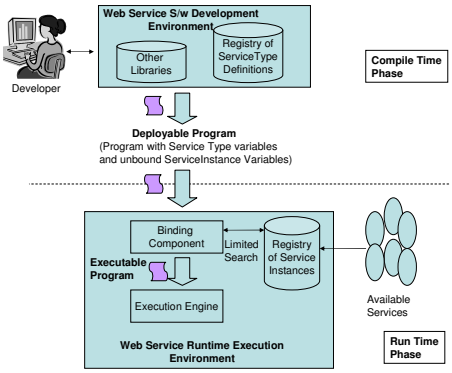

Figure 1: Programming Model for Services

Figure 1 shows the resulting programming model for service oriented software development that enables to split the development process across design time and runtime phases. This is essential to shield the developers from the dynamics of runtime environment.

As shown in the figure, the development environment provides a service types registry using which the developers could program to service interfaces without worrying about live instances. These service oriented programs could then be deployed by binding the service variables to actual running instances available at that time. The runtime execution environment provides access to instances registry.

To utilize this programming model, we introduce two new language constructs namely **ServiceType** and **ServiceInstance**. ServiceTypes and ServiceInstances are analogous to Classes and Objects, respectively, yet different. While ServiceTypes refer to definition of a service just like a class defines an entity and ServiceInstance refers to a live service similar to objects, the difference lies in the fact that both ServiceTypes and ServiceInstances are first class entities. Classes are typically not first class entities such as in C++. The ServiceType variables can be used to encode logic applicable at compile time while ServiceInstance variables allow encoding of logic applicable at runtime.

They could be declaratively defined as partially shown in

WWW 2008, April 21–25, 2008, Beijing, China.
ACM 978-1-60558-085-2/08/04.

```
ServiceType{
  functionalSpec{
       interfaceTypeA{
           inputTypes { .. };
           outputTypes { .. };
           preconditions { .. };
           effects { .. };
       };
       ….
  };
  modelDescription{ .. };
  stateDescription {
    <List of types of data
     members>
  };
  nonFunctionalReqmts{
        <QoS and other reqmts.>
  };
}
```

```
ServiceInstance{
  serviceTypeRef;
  operationalSpec{
     interfaceA{
       inputs {..};
       outputs{..};
     }
     ….
  }
  state{
   <list of data variables>
  }
  nonFunctionalCapability{
     <QoS guarantees>
  }
  binding{
    <list of protocols,ports>
  }
}
```

Figure 2: *ServiceType and ServiceInstance*

Fig. 2. Alternatively, ServiceType could be defined with the help of service type descriptions available in the types registry. The definition in such a case would take the form:

```
new ServiceType(XMLFileRef FS, XMLFileRef MD, XMLFileRef SD, XMLFileRef NFR)
```

ServiceType construct is composed of four elements - a functional specification containing various interface descriptions, a model description, description of State in terms of a list of data types, and constraints on non-functional capabilities of service instances. Similarly, ServiceInstance could also be defined with the help of service instance descriptions in instances registry as given below:

```
new ServiceInstance(serviceTypeRef, XMLFileRef OS,
XMLFileRef State, XMLFileRef NFC, XMLFileRef binding)
```

The 'new' operation here creates a new ServiceInstance entity and binds it to an actual running service instance, in the services registry, whose service details are provided through the XMLFile references in the parameters.

The functional specification in the ServiceType definition enables the software developers to write logic that reasons upon the functionality of the service they are dealing with. The operational specification in ServiceInstances allows logic that invokes the service's functionality.

Operations on Service Types are defined based upon the notion of classification, composition and inheritance defined for services in [2],

-Classification: *isEquivalentTo(ST)* returns a degree of match between the current service type and the service type supplied as parameter; *isBound()* returns true if a service type has been instantiated or bound to at least one instance; *getInstance(ST)* returns an existing bound ServiceInstance or creates a new binding to an actual running service instance and returns it as a ServiceInstance; *isServiceTypeOf(SI)* returns a value indicating a degree of match i.e. direct type of the service instance vs a super type in chain.

- Composition: *isComposedOf(ListofSTs)* and *isComponentOf(CompositeST)* return a true if the current service type is composed of the supplied list of service types and if it is a component of the supplied service type respectively.

- Inheritance: *isSuperTypeOf(ST)* and *isSubTypeOf(ST)* return true if the current service type is a supertype and if it is a subtype of the specified service type respectively.

Operations on Service Instance variables include the functional methods (i.e. actual functionality offered by the instance) and others operations that are related to querying the interface descriptions or sending commands to the instance and verifying the health of the instance by invoking their management interfaces. The functional methods are invoked by referring to the interface names (such as interfaceA in Fig. 2) through the ServiceInstance variable. Queries for interface descriptions are supported by invoking an appropriate operation on the associated ServiceType. Such operations on instances would help automatic agents to programmatically interpret and subsequently invoke the interface of new services. The management commands and health check operations are dependent upon the management interfaces implemented by the instances.

3. RELATED WORK

ServiceJ [4] proposes extensions to Java to enable support for service oriented computing in OO languages. Declarative language constructs are proposed to transparently deal with service selection at runtime as well as for applying a filtering and ranking criteria. However, their extensions deal primarily with non-functional aspects and it is not clear how their compiler works without having access to functional description of services available at compile time. The ServiceType and ServiceInstance constructs proposed in this paper address the functional as well as non-functional aspects of services in a unifying development framework. Zimmermann et al. [6] motivate the need for a Service Oriented Analysis and Design (SOAD) approach that leverages and builds upon existing approaches of Object Oriented Analysis and Design (OOAD), Enterprise Architecture frameworks and Business Process Modeling concepts. Papazoglou [5] provides a detailed comparison between services in SOA and objects in OOAD. However, they felt that concepts like polymorphism etc. are not applicable to SOA. The language constructs proposed by us enable such OO features by raising the programming abstraction from objects to services.

4. CONCLUSION

We have proposed programming language constructs to enable programming with services. The constructs incorporate well established object oriented software development principles such as classification, composition and inheritance into a service oriented programming model. We are currently in the process of developing a compiler for introducing these constructs as extensions to Java.

5. REFERENCES

[1] V. Agarwal, K. Dasgupta, N. Karnik, A. Kumar, A. Kundu, S. Mittal, and B. Srivastava. A Service Creation Environment based on End to End Composition of Web Services. In *Proceedings of WWW*, May 2005.

[2] A. Kumar, A. Neogi, and D. J. Ram. An OO Based Semantic Model for Service Oriented Computing. In *Proc. of IEEE SCC, USA*, Sept. 2006.

[3] A. Kumar, S. Pragalapati, A. Neogi, and D. Janakiram. Raising Programming Abstraction from Objects to Services. In *Proceedings of ICWS*, July 2007.

[4] S. D. Labey, M. van Dooren, and E. Steegmans. ServiceJ: - A Java Extension for Programming Web Service Interaction. In *Proc. of IEEE ICWS*, Jul 2007.

[5] M. P. Papazoglou. Service-Oriented Computing: Concepts, Characteristics and Directions. In *Proc. of WISE*, Dec 2003.

[6] O. Zimmermann, P. Krogdahl, and C. Gee. Elements of Service-Oriented Analysis and Design: An interdisciplinary modeling approach for SOA project. *http://www.ibm.com/developerworks/webservices/library/ws-soad1/*, June 2004.