Extending Enterprise Services Descriptive Metadata with Semantic Aspect Based on RDF

Lei Zhang, Yani Yan and Jianlin Wu

Beijing Key Laboratory of Intelligent Communications Software and Multimedia, Beijing University of Posts and Telecommunications, Beijing 100876, P.R. China
zlei@bupt.edu.cn kimsky61@gmail.com jlwu@bupt.edu.cn

Abstract. In the area of enterprise information integration, enterprise services are usually pre-defined and constructed before being supplied to users traditionally. However, with the rising of users' individuation needs, constructing and offering every user with the very service that can meet their needs by service provider is impossible and costly. In order to solve this problem, a new method of dynamic constructing and offering services defined by users themselves is put forward in this paper. Firstly, the framework of the service integration system is illustrated. Then, a formalized definition of enterprise services is brought forward. Based on the definition, a meta-model extended with semantic is constructed by RDF (Resource Describe Framework). Metadata defined based on the meta-model can describe the constructive components of a service and their communications. Users can define specific service by subscribing service descriptive metadata. After subscription, services can be called and system will parse the service metadata and dynamic create the service according to the service structure and components operation logic specified in the metadata. At last, a scenario is provided to demonstrate how the meta-model can be used to support the dynamic component-based construction of services.

Keywords: *Component-oriented architecture, Enterprise information integration*

l. INTRODUCTION

Every enterprise faces the challenge of integrating components of diverse IT systems. Components are the basic building blocks of enterprise and distributed applications [1]. How to integrate these components together to dynamic create services that can be offered to enterprise customers to make use of existent enterprise resources furthest and to make enterprise gain from it? The post facto integration of independent components has not been particularly successful. [2]Lack of agreed upon technological standards and the absence of semantic integration models are two major contributing factors [2].

The dominating component standards including Common Object Request Broker Architecture(CORBA), Component Object Model (COM), Enterprise Java Beans (EJB) etc.. CORBA is a communication Middleware that can separate the applications from communication details. CORBA [3] can offer components of distributed systems through consistent interface specification. EJB [3] emphasize on reuse of components.

EJB offers component operating environment to make business developers can concentrate on developing business applications [4]. However, they are inefficient in dynamic integration of components [4] because they are inefficient in describing components behavior semantic [5].

Web Service Business Process Execution Language (WS-BPEL) [6] is an XML-based standard for modeling business process flows within the Web services architecture [7]. WS-BPEL can effect as a component integration standard. However, WS-BPEL can only be used to integrate Web services components into business process [6]. Other resources, like EJB or COM. Components have to be encapsulated into Web services to be integrated.

Other component integration and dynamic service composition systems lack extensibility and understandability and are inefficient in service specialization. Also some systems cannot support operation dependency and soft-state information. With the rising of users' individuation needs, constructing and offering every user with the very service that can meet their needs by service provider merely is impossible and costly. In order to solve this problem, a new method of dynamic integrating components to construct services based on the component integration model specified in service descriptive metadata subscribed by users is put forward in this paper. Services have not to be constructed before being supplied to users and they can meet users' needs furthest as they are defined by users themselves. The new method can provide several benefits: flexibility, adaptability, and availability.

Section 2 will introduce some background knowledge. The framework of the service integration system is illustrated in section 3. A formalized definition of services is described in section 4 and the service specification meta-model is built by RDF based on the formalization in section 5. At last, in section 6, an example on how to build metadata for specific service based on the meta-model is given.

2. BACKGROUND KNOWLEDGE

2.1 Metadata

Metadata has traditionally been defined as "data about data" or "information about information". Metadata was invented to help computer systems and humans more efficiently and effectively organize, access, and interpret data. [8] Metadata is about knowledge, which is the ability to turn information and data into effective action. Meta-model supplies the template for constructing metadata. Meta-model is simply the elements that are used to describe the data, information and knowledge.

In this paper, we will define the dynamic service descriptive meta-model. Metadata based on the meta-model contains semantic information which describes the dynamic service, including components that constitute the service and their operation sequence.

2.2 RDF

The Resource Description Framework (RDF) is a language for representing information about resources in the World Wide Web [9]. RDF is intended for situations in which this information needs to be processed by applications, rather than being only displayed to people [9]. RDF provides a common framework for expressing this information and application designers can leverage the availability of common RDF parsers and processing tools.

RDF is based on the idea of identifying things using Web identifiers (called Uniform Resource Identifiers, or URIs [9]), and describing resources in terms of simple properties and property values. RDF provides an XML-based syntax (called RDF/XML[9]) for recording and exchanging information.

RDF is particularly intended for representing metadata about Web resources [9]. RDF provides a way to express simple statements about resources. However, RDF user communities also need the ability to define the vocabularies they intend to use, specifically, to indicate that they are describing specific kinds of resources, and will use specific properties in describing those resources. RDF itself provides no means for defining such application-specific classes and properties [9]. Instead, such classes and properties are described as an RDF vocabulary, using extensions to RDF provided by the RDF Vocabulary Description Language 1.0: RDF Schema [9]. RDF Schema does not provide a vocabulary of application-specific classes.

In this paper, we will define our service-specific RDF vocabulary, including RDF classes and properties that can be used to construct the dynamic service descriptive metadata.

3. FRAMEWORK

The service management system needs to be setup. Through the system, user can submit service descriptive metadata and call the service after subscription. And the system will dynamic parse the service metadata and offer service to users by executing related components according to the execution logic specified in the metadata.

Figure1 below will show the framework of the enterprise service management system.

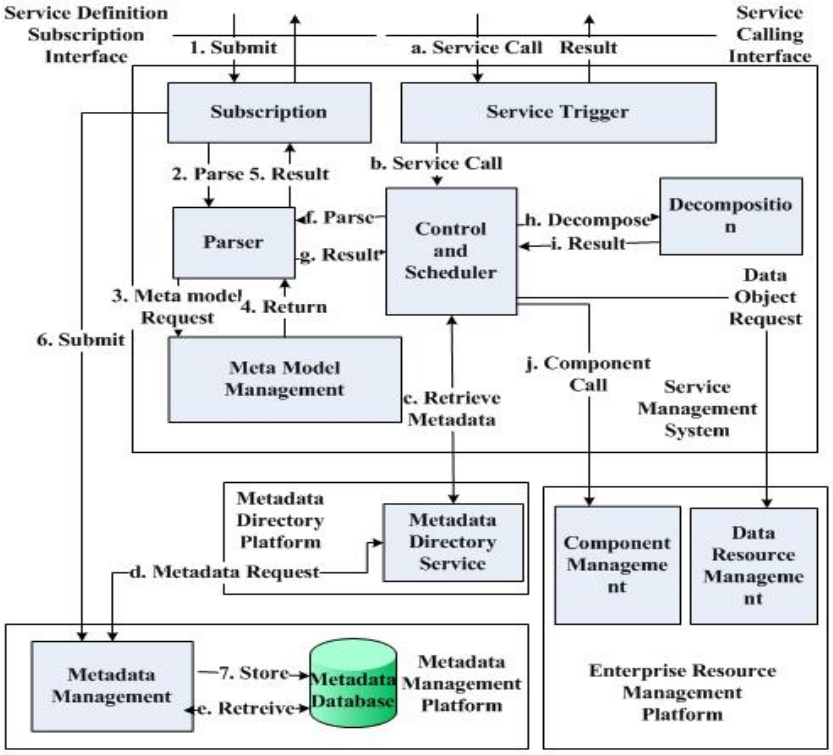

Figure 1. Framework of Enterprise Service Management System

Subscription module is the interface for subscribing service metadata. It first communicates with the Parse module to parse and validate the metadata before subscribing it to the metadata management platform. The Parse module will request meta-model from the Meta-model Management module to parse metadata based on the meta-model. At last, metadata will be stored to the Metadata DB by the Metadata Management module. Messages marked from 1 to 7 describe the procedure.

Service Trigger module is the interface for calling services. It will deliver service requests to the Control and Scheduler module which is the central controller for executing services. The Control and Scheduler module first requests the service metadata through the Metadata Directory service and then call the Parser module to parse it in order to understand the service operation logic. Then, the Decomposition module will be called by the controller to decompose the service into smaller tasks. During executing the tasks, the controller will communicates with the Enterprise Resource Management Platform to utilize related system components and access related data resource. Messages marked from a to j describes the procedure.

The Parser, the Control and Scheduler, the Decomposition and the Meta Model Management modules are under development. The following sections of this paper will concentrate on describing how to construct the service meta-model based on RDF and how to build service metadata based on the meta-model.

4. FORMALIZATION

This section will give a formalized definition of enterprise services. Services can be formalized as: Service (SID, CD, SIFD, CPC, ISC, DOC, PC). SID refers to Service ID, which is the universal service identity within the enterprise. CD refers to Context Description.

CD can be formalized as: CD (FD, UD, ID, RAP). FD refers to Function Description. FD describes the functional characters of services; UD refers to User Description and UD describes users who have privilege of calling the service; ID refers to Environment Description and ID describes the desirable executing environment of the service; RAP refers to Resource Access Privilege.

SIFD refers to Service Interface Description. SIFD describes the service interface through which service can be called by users.

CPC refers to Component Collection. CPC can be formalized as: CPC (CPID).CPID refers to Component ID. Furthermore, component can be formalized as: C(CPID, CPD, AC). CPD represents Component Description and AC represents collection of actions that constitute the component.

ISC represents Inter Sequence of Components, which can be formalized as: ISC((Action1, Action2), (Time1 (Action1), Time2 (Action2), ST)). ISC is comprised of an action collection and an expression which explains the temporal executing sequence of actions included in the collection. Action represents dynamic executing of specific component function. Action can be formalized as: Action (CPID). CPID indicates the component to which the action belongs. Time represents the states of execution of actions. There are two types of states: Begin and End. Begin indicates the start point of an action and End indicates the ending point of the action. ST refers to Sequence of Time. ST indicates the temporal relationship between two action Times and there are three types of ST: Before, After and Simultaneity. For example, the ISC expression ((Action1, Action2), (End (Action1), Begin (Action2), Before)) indicates that Action2 can only be triggered after Action1 has reached its ending point. So, Before means that the former occurs before the latter and After means the reverse against Before as well as Simultaneous means synchronous.

DOC represents Data Objects Collection. Data Objects are logical data resource within the enterprise that can be accessed by services. DOC can be formalized as: DOC (DOID). DOID represents Data Object ID, which is the universal identity within the enterprise. Data Object can be formalized as: DO (DOID, DOD, PDC). DOD represents Data Object Description. DOD describes the owner, access privilege, update time and other characteristics of data objects. PDC represents Physical Data Collection, which indicates the physical data storage to which the logical data object can be mapped. We assume that one logical data object can be mapped to several physical data storages which have the similar data structure. For example, we can define a train ticket sales data object and map it to train ticket sales data of different provinces. Train ticket sales data of different provinces are different physical data storages. Thus, PDS which represents Physical Data Storage can be formalized as:

PDS (PDID, PDD, L, DSD). PDID is the universal identity within the enterprise and PDD represents Physical Data Description, which describes the owner, access

privilege, update time and other characteristics of the data storage. L indicates the physical location of the data storage and DSD represents Data Structure Description, which describes the physical structure of the data storage. PC represents Parameter Collection. The following sections will explain how to construct meta-model of dynamic services.

5. SERVICE SPECIFIC RDF VOCABULARY

Meta-model offers the template for defining metadata. Meta-model need to be built to make service descriptive metadata constructed according to conformable criterion that can be comprehended by the service executing system.

In this paper, we will construct the service-specific RDF vocabulary to setup the service meta-model. The service-specific RDF vocabulary includes definition of RDF classes and properties.

5.1 Definition of RDF Classes

Figure 2 shows part of the definition:

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF
   Xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   Xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   Xmlns:base= "http://service/metadata/rdf-schema#">
<rdfs:Class rdf:ID="Service"/>
<rdfs:Class rdf:ID="Component"/>
<rdfs:Class rdf:ID="DataObject"/>
<rdfs:Class rdf:ID="PhysicalData"/>
<rdfs:Class rdf:ID="Parameter"/>
<rdfs:Class rdf:ID="Action"/>
<rdfs:Class rdf:ID="InterSequence"/>
<rdfs:Class rdf:ID="TimeSequence"/>
</rdfs:RDF>
```

Figure 2. Definition of RDF Classes

Definition of class Action is similar to the formalized definition of Action described in section 4. Action represents dynamic executing of specific function of specific component. Definition of class InterSequence is similar to the formalized definition of ISC ((Action1, Action2), (Time1(Action1), Time2(Action2), ST)) described in section 4. It describes the executing sequence of different actions. Object of class TimeSequence represents the (Time1(Action1), Time2(Action2), ST) part of object of class InterSequence.

5.2 Definition of RDF Properties

Figure 3 shows part of the definition:

```
<rdf:Property rdf:ID="serviceID"> <rdfs:domain rdf:resource="#Service"/> </rdf:Property>
<rdf:Property rdf:ID="componentCollection"> <rdfs:domain rdf:resource="#Service"/> </rdf:Property>
<rdf:Property rdf:ID="dataObjectCollection"> <rdfs:domain rdf:resource="#Service"/> </rdf:Property>
<rdf:Property rdf:ID="interSquenceCollection"> <rdfs:domain rdf:resource="#Service"/> </rdf:Property>

<rdf:Property rdf:ID="componentID"> <rdfs:range rdf:resource="&xsd;integer"/> </rdf:Property>
<rdf:Property rdf:ID="memberAction"> <rdfs:domain rdf:resource="#Component"/> </rdf:Property>

<rdf:Property rdf:ID="actionCollection"></rdf:Property>
<rdf:Property rdf:ID="timeSequenceCollection"></rdf:Property>

<rdf:Property rdf:ID="actionTime">
   <rdfs:domain rdf:resource="#TimeSequence"/>
   <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
<rdf:Property rdf:ID="sequenceType">
   <rdfs:domain rdf:resource="#TimeSequence"/>
   <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>
```

Figure 3. Definition of RDF Properties

6. EXAMPLE AND SCENARIO

The service-specific RDF vocabulary can effect as the service meta-model for defining service descriptive metadata. An example on how to build service descriptive metadata based on the service-specific RDF vocabulary is given below:

Suppose that enterprise A offers data mining components which can be utilized by users to define data mining services that can meet their needs. Some of the components can be combined to offer classification and forecast service which can be used for customers chum prediction, credit card fraud identification, heart disease diagnose etc.. In order to build the service, three kinds of components: Data Reader component, Classification Modeling component and Classification Forecast component are essential and the Discretization component is optional.

Data Reader component reads data from files and prepare it for constructing classification and forecast model. Classification Modeling component is responsible for building and validating the model based on classified historical data and Classification Forecast component uses the model built to forecast the attributive class of unclassified data. Discretization component disperses continuous data attributes to discrete values because only discrete data attributes can be used to build classification model. Discretization component offers two kinds of discretization: discretization according to specific standard and discretization by auto-learning.

Users may need different classification and forecast services. E.g. customer B who wants to identify credit card fraud may choose to use standard discretization function

of Discretization component and essential components to build the service because some attributes of the consumption data are continuous. Whereas customer C who wants to predict customer chum may like to combine the auto-learning discretization function of the Discretization component and essential components to build the service because auto-learning discretization can produce more accurate result. Besides, customer D who wants to diagnose heart disease doesn't need the Discretization component at all because data of symptom is already discrete. Customer B, C and D can define services that can meet their needs by constructing service specification metadata based on the RDF vocabulary defined in section 5.

```
<rdf:RDF
   Xmlns:base= "http://service/metadata/rdf-schema# ">
   Xmlns:ex="http://service/metadata/ex#">
  <ex:Service rdf:ID= "classificationAndForecast">
     <rdfs:comment>Defined by user to offer classification and forecast service</rdfs:comment>
     <ex:componentCollection rdf:parseType="Collection">
     <rdf:Description rdf:about="http://service/metadata/ex#dataReader"/>
     <rdf:Description rdf:about="http://service/metadata/ex#discretization"/>
     <rdf:Description rdf:about="http://service/metadata/ex#classificationModeling"/>
     <rdf:Description rdf:about="http://service/metadata/ex#classificationForecast"/>
  </ex:componentCollection>
  <ex:interSquenceCollection rdf:parseType="Collection">
     <rdf:Description rdf:about="http://service/metadata/ex#classificationAndForecastInterSequence"/>
  </ex:interSquenceCollection>
</rdf:RDF>
```

Figure 4. RDF Definition of Service of Classification and Forecast

```
<ex:Component rdf:ID= "discretization">
    <rdfs:comment>Disperse continuous data into discrete value for modeling</rdfs:comment>
    <ex:componentID rdf:datatype= "&xsd;integer">2628</ex:componentID>
    <ex:memberAction rdf:resource="http://service/metadata/ex#autoLearningDiscretizationAction"/>
</ex:Component>
<ex:Component rdf:ID= "classificationModeling">
    <rdfs:comment>Build and validate classification model based on historical training data</rdfs:comment>
    <ex:componentID rdf:datatype= "&xsd;integer">2629</ex:componentID>
    <ex:memberAction rdf:resource="http://service/metadata/ex#classificationModelBuildAction"/>
    <ex:memberAction rdf:resource="http://service/metadata/ex#classificationModelValidationAction"/>
</ex:Component>
```

Figure 5. RDF Definition of Service Component

```
    <ex:Action rdf:ID= "autoLearningDiscretizationAction">
        <ex:actionID rdf:datatype= "&xsd;integer">262801</ex:actionID>
        <ex:componentID rdf:datatype= "&xsd;integer">2628</ex:componentID>
    </ex:Action>
    <ex:Action rdf:ID= "classificationModelBuildAction">
        <ex:actionID rdf:datatype= "&xsd;integer">262901</ex:actionID>
        <ex:componentID rdf:datatype= "&xsd;integer">2629</ex:componentID>
    </ex:Action>
    <ex:Action rdf:ID= "classificationModelValidationAction">
        <ex:actionID rdf:datatype= "&xsd;integer">262902</ex:actionID>
        <ex:componentID rdf:datatype= "&xsd;integer">2629</ex:componentID>
    </ex:Action>
    <ex:Action rdf:ID= "forecastAction">
        <ex:actionID rdf:datatype= "&xsd;integer">263001</ex:actionID>
    </ex:Action>
```

Figure 6. RDF Definition of Service Action

```
<ex:InterSequence rdf:ID= "classificationAndForecastInterSequence">
   <ex:actionCollection rdf:parseType="Collection">
     <rdf:Description rdf:about="http://service/metadata/ex#autoLearningDiscretizationAction"/>
     <rdf:Description rdf:about="http://service/metadata/ex#classificationModelBuildAction"/>
     <rdf:Description rdf:about="http://service/metadata/ex#classificationModelValidationAction"/>
     <rdf:Description rdf:about="http://service/metadata/ex#forecastAction"/>
   </ex:actionCollection>
   <ex:TimeSequenceCollection rdf:parseType="Collection">
     <rdf:Description rdf:about="http://service/metadata/ex#discretizationAndBuildTimeSequence"/>
     <rdf:Description rdf:about="http://service/metadata/ex#validationAndForecastTimeSequence"/>
   </ex:TimeSequenceCollection>
</ex:InterSequence>
```

Figure 7. RDF Definition of Service InterSquence

```
<ex:TimeSequence rdf:ID= "discretizationAndBuildTimeSequence">
   <ex:actionTime rdf:parseType=" Resource"
     <ex:keyword rdf:datatype= "&xsd;string">end</ex:keyword>
     <ex:actionID rdf:datatype= "&xsd;integer">262801</ex:actionID>
   </ex:actionTime>
   <ex:actionTime rdf:parseType="Resource"
     <ex:keyword rdf:datatype= "&xsd;string">begin</ex:keyword>
     <ex:actionID rdf:datatype= "&xsd;integer">262901</ex:actionID>
   </ex:actionTime>
   <ex:sequenceType rdf:datatype= "&xsd;string">before</ex:sequenceType>
</ex:TimeSequence>
<ex:TimeSequence rdf:ID= "validationAndForecastTimeSequence">
   <ex:actionTime rdf:parseType=" Resource"
     <ex:keyword rdf:datatype= "&xsd;string">begin</ex:keyword>
     <ex:actionID rdf:datatype= "&xsd;integer">263001</ex:actionID>
   </ex:actionTime>
   <ex:actionTime rdf:parseType="Resource"
     <ex:keyword rdf:datatype= "&xsd;string">begin</ex:keyword>
     <ex:actionID rdf:datatype= "&xsd;integer">262902</ex:actionID>
   </ex:actionTime>
   <ex:sequenceType rdf:datatype= "&xsd;string">simultaneous</ex:sequenceType>
</ex:TimeSequence>
```

Figure 8. RDF Definition of Service TimeSequence

The above figure 4 to 8 show part of the definition of the service metadata defined by customer C who wants to predict customer chum.

Definition of timeSequence “discretizationAndBuildTimeSequence” indicates that action “classificationModelBuildAction(actionID=262901)” can only be triggered after action “autoLearningDiscretizationAction(actionID=262801)” has finished executing.Definition of timeSequence “ValidationAndForecastTimeSequence” indicates that action “classificationModelValidationAction(actionID=262902)” and action “forecastAction(actionID=263001)” can execute concurrently.

Users can subscribe the service descriptive metadata to the service system. Since subscription, once the service is called, system will load and parse the service metadata and dynamic create the service according to the service structure and components operation logic specified in the metadata.

7. CONCLUSION AND FUTURE WORK

A new method of dynamic offering user-defined services was put forward in this paper. Firstly, the framework of the service management system was illustrated. Then, in order to let user define services that can meet their needs, the service-specific RDF vocabulary was built to define service meta-model based on which service descriptive metadata can be constructed. At last, an example was given.

In the future, we will keep on working at development of the system core modules. At the same time, we will concentrate on analysis of service performance, model validation and design of transaction management mechanism.

REFERENCES

1. L. Roger, H. Ashok, C.-C. Chiang , H.-S. Yang, and H.-K. Kim, A framework for dynamically converting components to web services, in *Proc. of Third ACIS International Conference on Software Engineering Research, Management and Applications, SERA 2005,* eds. R. Lee, K. W. Lee, and B. Malloy (IEEE Computer Society: Piscataway, NJ, 2005), pp.431-437.
2. J. Leon, Towards semantic integration of components using a service-based architecture, *Journal of Integrated Design and Process Science*. Volume 9, Number 3, pp.1-13, (2005).
3. S. Michael, CORBA 3, in *Proc. of the 34th International Conference on Technology of Object-Oriented Languages and Systems, TOOLS 34*, eds. Q. Li, D. Firesmith, R. Riehle, G. Pour, and B. Mayer (IEEE Computer Society: Piscataway, NJ, 2000), pp.397.
4. E. Wolfgang and K. Nima, Component technologies: Java Beans, COM, CORBA, RMI, EJB and the CORBA component model, in *Proc. of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering,* eds. Volker Gruhn (ACM Press: New York, NY, 2001), pp.311-312.
5. Y. Wu, K. Zhang, X. Wang, J. Tian, and Y. Chen, Extending Metadata with Semantic Aspect in Component-based Distributed System, *Computer Engineering.* Volume 32, Number 12, pp.68-70, (2006).
6. A. Arkin, S. Askary, B. Bloch, F. Curbera, Y. Goland, N. Kartha, C. Liu, S. Thatte, P. Yendluri, and A. Yiu, *Web Services Business Process Execution Language Version 2.0*, OASIS (2005). http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel (Accessed May 18, 2007).
7. Anonymous, *Web Services Architecture*, W3C (2004). http://www.w3.org/TR/ws-arch (Accessed May 18, 2007).
8. S. John and S. Peter, Metadata standards roundup, *IEEE Multimedia.* Volume 13, Number 2, pp.84-88, (2006).
9. Anonymous, *RDF Primer Recommendation*, W3C (2004). http://www.w3.org/TR/rdf-primer(Accessed May 18, 2007).