基于接口匹配的 Web 服务自动组合*)

于守健　何　丰　乐嘉锦
(东华大学计算机科学与技术学院　上海 200051)

摘　要　通过把 Web 服务抽象成为具有输入和输出接口的实体，提出了基于接口匹配的 Web 服务组合方法.以当前标准—WSDL(Web Services Description Language，简称 WSDL)作为服务接口的描述语言，从消息参数名和数据类型两方面通过使用聚类技术，将操作中的参数名归类为具有语义的概念，结合现有的 WordNet 语义词典，提出了服务接口匹配算法。该算法通过计算服务接口之间的相似程度，找出潜在的消息依赖关系，能够自动、动态地发现潜在的 Web 服务，从而实现 Web 服务自动组合。最后，在 TRP 系统上对匹配算法进行了实验评估。

关键词　Web 服务组合，接口匹配，聚类，WSDL

Automatic Web Service Composition Based on Interface Matching

YU Shou-Jian　HE Feng　LE Jia-Jin
(College of Computer Science and Technology, Donghua University, Shanghai 200051)

Abstract　This paper proposes a novel automatic Web service composition method based on interface matching, by abstracting Web service as an entity with input and output interface. With the current standard—WSDL as Web service description language, the interface matching algorithm is proposed from two aspects: name and data type of parameter in Web service message. This algorithm utilizes both clustering techinique and WordNet for parameter name matching. Parameters in service message are classified into concept cluster by utilizing clustering technique. The interface matching algorithm can discover potential message dependency by computing the similarity between service interface, which fulfils automatic and dynamic Web service discovery. Thus the automatic Web service composition is achieved. In the end, the matching algorithm is evaluated on the Textile Resource Planning system by experiments.

Keywords　Web service composition, Interface matching, Clustering, WSDL

1　引言

随着 Web 服务技术、规范的发展，越来越多的企业将自己的业务能力包装成 Web 服务发布。但是，单个 Web 服务的功能有限，难以满足企业级应用的流程集成需要。Web 服务组合能把相对简单的 Web 服务，按业务流程逻辑组合起来，从而提供更强大、更完整的业务功能[1]。目前，Web 服务组合是工业界和学术界研究的热点问题[2~4]。一方面，Web 服务组合与工作流技术很相似，因此工业界在当前敏捷工作流、自适应工作流和跨企业业务流程集成方面取得的成果的基础上，研究 Web 服务的自动组合[5,6]；另一方面，为了实现 Web 服务自动组合过程中的智能推理，学术界结合语义 Web 和 Web 服务，做了大量研究工作[7,8]。

Web 服务组合问题的关键是如何以最少的人工参与来提供组合服务。组合服务是由多个基本服务组成的，在其执行过程中，数据在各个基本服务之间进行传递，一个服务输出的数据作为另一个服务的输入。因此，服务之间由于所需要访问的数据而产生约束关系，这为 Web 服务组合提供了新的思路。服务的接口定义了服务之间的数据流信息，通过把 Web 服务抽象为具有输入和输出接口的实体，其接口的参数反映了 Web 服务操作的输入输出数据流，数据流则又反映了服务之间的消息依赖关系。因此，通过计算服务接口之间的相似程度，找出潜在的消息依赖关系，就能够自动、动态地发现潜在的 Web 服务，从而实现 Web 服务的自动组合。WSDL 是当前 Web 服务接口描述语言的标准[9]，各种业务流程建模语言，诸如 BPEL 和 WSCI 等，都是建立在 WSDL 的基础之上[10]。WSDL 以 XML 格式定义了 Web 服务的消息，这些 XML 消息是活动之间数据流的重要组成部分。因此本文以 WSDL 为接口描述语言，提出了基于接口匹配的 Web 服务自动组合方法。

本文首先介绍了 Web 服务接口描述的标准语言—WSDL；第 3 节介绍了接口参数聚类算法，将操作中的参数名归类为具有语义意义的概念；在此基础上，第 4 节提出了接口匹配算法；第 5 节介绍了基于接口匹配进行 Web 服务组合的过程；第 6 节对接口匹配算法进行了实验评估；最后是对本文的总结。

2　WSDL 结构分析

WSDL 定义了一套基于 XML 的语法，将 Web 服务描述为能够进行消息交换的服务访问点或端口(Port)的集合。在 WSDL 中，服务访问点和消息的抽象定义，从具体的服务部署或数据格式绑定中分离出来。消息是指对所交换数据的抽象描述；而端口类型是指操作的抽象集合。用于特定端口类型的具体协议和数据格式规范，构成了可以再次使用的绑定。

*)上海市科委产学研联盟项目(05DZ11C06)资助。于守健　博士，研究方向为企业应用集成、Web 服务、数据库与数据仓库；何　丰　博士生，研究方向为 Web 服务与企业应用集成；乐嘉锦　教授，博士生导师，研究方向为数据库、数据仓库、软件复用。

将 Web 访问地址与可再次使用的绑定相关联，可以定义一个端口，而端口的集合则定义为服务。可以参考图 1 理解 WSDL 的结构组织。

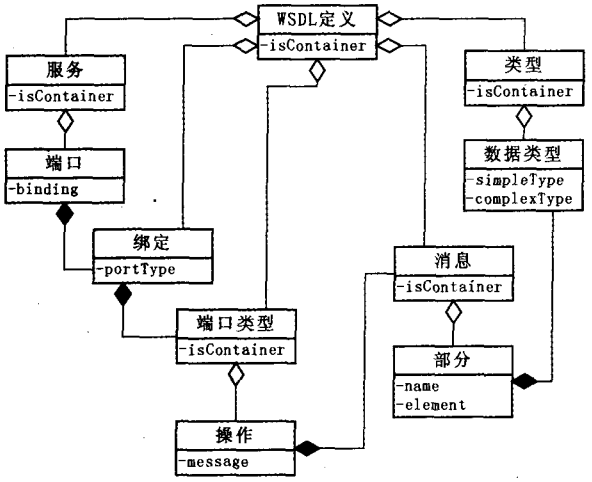

图 1　WSDL 的数据结构

Types、Message 和 PortType 三种结构描述了调用 Web 服务的抽象定义。其中，Types 是一个数据类型定义的容器，它包含了消息中所有 XML 元素类型的定义，通常是采用 XML Schema 进行类型定义。Message 具体定义了在通信中使用的消息的数据结构，其中包含了一组 Part 元素，每个 Part 元素都是最终消息的一个组成部分，并且都会引用 Types 中的一个 DataType 来表示它的结构。PortType 具体定义了一种服务访问入口的类型，即传入/传出消息的模式及其格式。一个 PortType 可以包含若干个 Operation，而一个 Operation 则是指访问入口支持的一种类型的调用。

一个 Web 服务可以用一个三元组(C,A,P)表示，如定义 1 所示。其中，C 是指服务的内容，也就是对 Web 服务的语义描述；A 是指服务提供的活动，也就是 Web 服务中的 operation；P 描述服务的一些属性，比如服务质量、费用等。C 和 P 属于服务的语义范畴。

定义 1(Web 服务)　一个服务表示为三元组(C,A,P)。任取 $a\in A$，a 为服务中的一个操作，$a=(op,I,O)$。op 表示操作名，I 表示 op 的输入消息参数，$I=(p_1,\cdots,p_m)$；O 表示 op 的输出消息参数，$O=(q_1,\cdots,q_n)$；其中 $p_i(i=1,\cdots,m)$和 $q_j(j=1,\cdots,n)$是形式为$\langle name\rangle:\langle type\rangle$的一个消息参数。

Web 服务提供的功能是通过调用其中的操作来完成的，因此，本文从服务提供的功能方面，研究 Web 服务操作层次上的自动组合。两个操作，当源操作的消息类型与目的操作的消息类型相匹配时，这两个操作可以顺序执行。这个约束限制了服务之间可能的连接，也为操作的组合提供了新的思路。每个 Web 服务有一个与之相关联的 WSDL 文档，描述服务的功能和接口。每个服务包含一系列的操作，每个操作有一个名字和相对应操作的输入和输出参数。WSDL 文档对每个参数的名字和数据类型都做了描述。如果能根据目的服务操作的输入输出消息，找出潜在的消息依赖关系，那么就可以实现目的组合 Web 服务。因此，基于接口匹配进行 Web 服务组合的过程，就是根据操作的输入输出参数，找出存在的消息依赖关系的过程。而这其中，接口匹配是关键。

3　Web 服务接口参数聚类

Web 服务接口匹配的难点在于，其接口输入输出参数的名字并不能完全表达操作的语义。因此，简单地根据操作名和输入输出参数名，不能实现有效的操作匹配。要有效匹配 Web 服务操作的输入输出，必须能处理它们潜在的语义，但是要做到这一点是很困难的。参数名由开发人员按他们的意愿来定义，往往使用不同的命名规则，或者不同的同义词定义同一个对象，参数名不一定由正确的英语单词组成，并且会存在错误的拼写或缩写词等。

一个参数名通常是由连成一串的单词组成，比如 *requestPrice* 由 *request* 和 *price* 两个单词组成，我们称组成参数的一个单词为项(*term*)。根据项在 Web 服务输入输出参数中的共同出现情况，将所有项聚类为有意义的概念(*concept*)簇，这样簇中的项就具有相同或相近的语义。在此基础上，再进行输入输出参数的匹配，能够更有效地发现相匹配的操作。本节讨论消息参数聚类算法。

3.1　组成参数术语的关联

我们的聚类算法是对聚集聚类的改进。从一个参数中可以拆解出多个项，例如 *requestPrice* 可拆解为 *request* 和 *price* 两个项。这些项同时出现在一个参数中，它们表达了相近的语义概念。因此，我们的聚类算法是基于一种表示项的共同出现的特殊关联规则，根据项在 Web 服务操作输入输出参数中出现的条件概率，对参数进行聚类。具体地讲，我们基于形式为 $t_1\rightarrow t_2(s,c)$的关联规则。其中，t_1、t_2 表示两个项，s 为支持度，$s=p(t_1)=\frac{\|IO_{t1}\|}{\|IO\|}$，表示 t_1 出现在一个输入/输出参数中的概率。$\|IO\|$ 表示操作中输入和输出参数的总数量。$\|IO_{t1}\|$ 表示含有 t_1 的输入和输出参数数量。c 为置信度，$c=p(t_2|t_1)=\frac{\|IO_{t1,t2}\|}{\|IO_{t1}\|}$，表示 t_1、t_2 同时出现在一个输入或输出参数中概率，$\|IO_{t1,t2}\|$ 表示既含有 t_1 又含有 t_2 的输入或输出参数的数量。$t_1\rightarrow t_2(s_{12},c_{12})$和 $t_2\rightarrow t_1(s_{21},c_{21})$有不同的支持度和置信度。

3.2　耦合度与相关度

好的聚类算法应使得概念内部参数的连接关系(内聚)尽可能强，即概念内聚度高；而概念间参数的相关性应尽可能弱，即概念间耦合度弱；并且出现频率极高或极少的参数不应当参加聚类，连接关系紧密的参数应该聚类到同一个概念中。在传统的聚类技术中，耦合度定义为簇中各点到簇中心欧几里得距离之和，簇间相关度定义为各簇中心点之间距离的和[11]。但是，因为本文所使用的度量维度的不同，该定义不能使用。在本文提出的关联规则的基础上，我们重新定义了耦合度和相关度。对于关联规则 $t_1\rightarrow t_2(s,c)$，如果置信度大于阈值 t_c，我们称 t_1 与 t_2 紧密关联。给定一个簇 I，定义 I 的耦合度为紧密关联的术语对占总术语对的百分比，用式(1)表示。显然，只有一个项的簇的耦合度为 1。

$$coh_I=\frac{\|\{i,j|i,j\in I,i\neq j,i\rightarrow j(c>t_c)\}\|}{\|I\|(\|I\|-1)} \tag{1}$$

给定两个簇 I 和 J，I 和 J 的相关度，定义为两个簇中具有紧密关联的术语对的百分比，用式(2)表示：

$$cor_{I,J}=\frac{C(I,J)+C(J,I)}{2\|I\|\|J\|} \tag{2}$$

其中 $C(I,J)=\|\{i,j|i\in I,j\in J,i\rightarrow j(c>t_c)\}\|$。

为了衡量聚类后得到的簇的质量，我们定义耦合度和相关度的比值，作为高耦合度和低相关度的折衷，如式(3)所示。我们的目的是得到高分数的比值，即簇内有高的耦合度，簇间有低的相关度。

$$score_C=\frac{\frac{\sum_{I\in C}coh_I}{\|C\|}}{\frac{\sum_{I,J\in C,I\neq J}cor_{I,J}}{\|C\|(\|C\|-1)/2}}=\frac{(\|C\|-1)\sum_{I\in C}coh_I}{2\sum_{I,J\in C,I\neq J}cor_{I,J}} \quad (3)$$

3.3 聚类算法

我们采用改进的凝聚聚类方法对术语聚类[12]。聚类过程首先把每个项本身初始化为一个簇，将所有的关联规则先以置信度，再以支持度降序排列，不考虑支持度小于一定阈值的关联规则。在每一步中，算法选择一个排序最前的关联规则，将这两个簇合并。触发簇 I 和簇 J 合并的条件为 $\exists i\in I, j\in J, i\rightarrow j(s>t_s, c>t_c)$。为了使簇的合并不减小簇的耦合度，我们定义一个更加严格的耦合条件：给定一个簇 I，如果一个项 t 与簇中至少一半的其它项有紧密关联关系，该项称为簇的核心项。我们规定：只有当合并后的簇中所有的项都是核心项时，两个簇才可以合并，如式(4)所示：

$$\forall i\in I\cup J, \|\{j|j\in I\cup J, i\neq j, i\rightarrow j(c>t_c)\}\| \geqslant\frac{1}{2}(\|I\|+\|J\|-1) \quad (4)$$

在参数聚类过程中，早期一个不合适的聚类将影响后续的聚类过程。虽然对聚类算法的合并过程附加了严格耦合条件的限制，但是这只能做到局部的优化，并不能做到全局优化。解决这个问题的方法是将现有的簇进行拆分，以得到耦合度更高的簇，拆分后的部分项参与合并。给定两个簇 I 和 J，$I'=\{i|i\in I, \|j|j\in I\cup J, i\rightarrow j(c>t_c)\|\geqslant\frac{1}{2}(\|I\|+\|J\|-1)\}$，$J'=\{j|j\in J, \|i|i\in I\cup J, j\rightarrow i(c>t_c)\|\geqslant\frac{1}{2}(\|I\|+\|J\|-1)\}$，$I'$ 表示簇 I 中与簇 I 和 J 中的项有紧密关联的项集合，J' 表示簇 J 中与簇 I 和 J 中的项有紧密关联的项集合，我们的算法根据以下规则决定是否拆分和合并：

(1)如果 $I'=I, J'=J$，那么 I 和 J 可以直接合并(如图2a)。

(2)如果 $I'\neq I, J'=J$，如图2(b)所示，直接合并I和J会违背耦合条件。有两种方法：(a)簇 I 拆分成 I' 和 $I-I'$，将 I' 与 J 合并；(b)不作拆分和合并。首先检查方法(a)合并后的簇是否满足耦合条件，如果不满足耦合条件，I 和 J 将不做拆分和合并，如果满足耦合条件，分别计算方法(a)和方法(b)耦合度与相关度的比值，选择比值高的方法。

(3)$J'\neq J, I'=I$ 的处理方法与(2)一样。

(4)如果 $I'\neq I, J'\neq J$，如图2(c)所示，直接合并 I 和 J 也是违背耦合条件。同样有两种方法：(a)将簇 I 拆分成 I' 和 $I-I'$，将簇 J 拆分成 J' 和 $J-J'$，合并 I' 和 J'；(b)不拆分也不合并。首先检查方法(a)合并后的簇是否满足耦合条件，如果不满足耦合条件，I 和 J 将不做拆分和合并，如果满足耦合条件，分别计算方法(a)和方法(b)的耦合度与相关度的比值，选择比值高的方法。

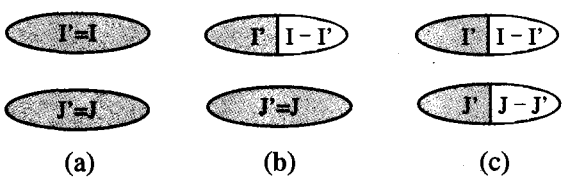

图2 簇的拆分与合并

经过以上的处理，合并的新簇满足耦合条件，但是拆分剩下的簇($I-I', J-J'$)未必满足耦合条件。为此，对这样的簇继续拆分出满足耦合条件的簇，直到所有的簇都满足耦合条件为止。

经过拆分、合并以后，在一个簇中仍然可能会有一些项与其它的项意义不相近，这样的项被称为噪音项。我们基于下面的方法检测噪音项，用 $\|IO_t\|$ 表示包含项 t 的输入/输出参数数量，用 $\|SIO_t\|$ 表示不包含 t 所在簇中其它任何一项的输入/输出参数数量，如果 $\|SIO_t\|\geqslant 1/2\|IO_t\|$，我们称 t 为噪音项。聚类算法执行一次聚类后，对概念簇结果进行一次扫描，以去除噪音项。算法1说明了参数聚类的整个过程。

	算法1//参数聚类
1	ClusteringParameters(T ;R)return (C){//T 表示项集，R 表示关联规则集，C 表示概念簇，聚类算法返回的结果
2	for (i=1; n)Ci={ti};//初始化概念簇，每个项作为一个簇
3	sort R;//对关联规则 R 先以置信度再以支持度降序排列
4	for each (r:$t1\rightarrow t2$(s >ts;c >tc)in R){
5	if t1 and t2 are in different clusters I and J
6	Compute I' and J';//计算 I'和J'，决定是否拆分与合并
7	if ($I'=I$·$J'=J$)merge I and J;//如图2 a，合并 I 和J
8	else if ()//如果拆分、合并后的簇满足耦合条件且具有较高耦合度、相关度比值，就进行拆分与合并
9	split and merge;
10	if()//如果拆分剩下的簇不满足耦合条件，继续拆分与合并过程
11	split I'' and/or J'' iteratively;}
12	scan inputs/outputs and remove noise terms; //去除噪音项
13	return result clusters;}//返回聚类参数簇

4 Web服务接口匹配

接口匹配不是一个新的概念，它在软件构件检索领域已得到广泛的应用。从构件中抽取出的特征信息称为签名，检索时把用户的检索要求和构件的签名相匹配，即签名匹配，这种方法首先是由 Zaremski 和 Wing 最先提出的[13,14]。签名匹配是构件检索的一种重要方法，但是不同构件检索系统对签名定义的形式不同[15]，因此，签名匹配方法不能直接应用于Web服务的接口匹配。WSDL已成为Web服务接口描述的工业标准，其语法的定义是基于XML模式的。第2节分析了WSDL的结构和语法，在此基础上，本节讨论基于WSDL结构和语法的接口匹配算法。

每个Web服务可以包含多项功能，每项功能对应一个操作，操作的输入消息和输出消息定义了Web服务的接口。在WSDL文档中，消息参数的数据类型用〈types〉元素定义，一个〈message〉元素是由多个〈types〉中的元素组成的，操作又是由多个消息组成的，如〈input〉，〈output〉和〈fault〉。两个消息的相似度取决于消息参数列表的相似程度，WSDL描述的数据类型是XML元素，XML元素的类型可以是原始数据类型、简单数据类型或复杂数据类型。原始数据类型是内置的可以直接使用的数据类型，如字符型、布尔型、数值型。简单数据类型和复杂数据类型是用户通过模式语言定义的。我们利用WSDL文档已经提供的信息，进行Web服务接口匹配，这实际上是通过匹配操作的输入和输出消息来实现的，也就是比较这些消息的数据类型。因此，本节首先讨论简单参数类型和复杂参数类型匹配，在此基础上提出了Web服务接口匹配的算法。

4.1 Web服务接口描述举例

```
<definitions> <types> <schema .... >
<element name="id" type="string"/>  <!--接口参数类型定义-->
<element name="name" type="string"/>  <!--简单参数类型-->
<element name="items">  <!--复杂参数类型-->
        <complexType>
                <all>
                        <element name="item" type="tns:Item"
                                minOccurs="0" maxOccurs="unbounded"/>
                </all>
        </complexType>
</element>
<complexType name="Item">
<all>
        <element name="quantity" type="int"/>
        <element name="product" type="string"/>
</all>
</complexType>
</schema> </types>
<message name="getDataRequest">  <!--操作的消息定义-->
        <part name="id" type="string"/>
</message>
<message name="getDataResponse">
        <part name="id" type="id"/>
        <part name="name" type="name"/>
        <part name="items" type="items"/>
</message>
<portType name="Data_PortType">
        <operation name="getData">  <!--操作定义-->
                <input message="getDataRequest"/>
                <output message="getDataResponse"/>
        </operation>
</portType> </definitions>
```

(a) 源服务 WSDL 文档举例

```
<definitions> <types> <schema .... >
<element name="id" type="int"/>
<element name="name" type="string"/>
<element name="price" type="float"/>
<element name="part" type="productParts"/>
<complexType name="productParts">
<all>
        <element name="productName" type="string"/>
        <element name="amount" type="int"/>
</all>
</complexType>
</schema> </types>
<message name="getProductRequest">
        <part name="id" type="int"/>
</message>
<message name="getProductResponse">
        <part name="id" type="id"/>
        <part name="name" type="name"/>
        <part name="price" type="price"/>
        <part name="part" type="part"/>
</message>
<portType name="Product_PortType">
        <operation name="getProduct">
                <input message="getProductRequest"/>
                <output message="getProductResponse"/>
        </operation>
</portType>  </definitions>
```

(b) 目的服务 WSDL 文档举例

图 3

为了说明 Web 服务接口匹配算法，本节举两个 WSDL 文档的例子，一个用于匹配过程中源服务接口的描述，一个用于目的服务接口描述，如图 3(a)和(b)所示，源服务包含一个操作 *getData*，目的服务包含一个操作 *getProduct*。

在上面的例子中，所有的操作都包含在标签〈portType〉中，每个操作以〈operation〉标注，在图 3 中以加粗短划线表示。操作的输入输出消息包含在标签〈message〉中，以加粗实线表示。数据类型包含在〈types〉中，以实线框表示。复杂数据类型以〈complexType〉标注，图 3(a)中的 *Item* 就是一个复杂数据类型，它包含两个简单数据类型 *product* 和 *quantity*。简单数据类型匹配是复杂数据类型匹配的基础，接下来本节讨论简单类型匹配算法。

4.2 简单参数类型匹配

Web 服务操作的输入输出参数类型都是以 XML Schema 定义的，其中的类型定义元素可分为简单类型或复杂类型。一个类型定义了一个参数，types 元素包含了所有消息中用到的 XML 元素。根据 XML Schema 的定义，简单数据类型是由元素名和值的类型属性确定的，因此，每个参数都表示为〈name〉:〈type〉的形式。元素名描述了元素的实际语义，值的类型属性是一个原始的数据类型。因此，简单数据类型匹配可以从元素名和值的类型属性两方面进行匹配。

对于元素名称匹配，我们可以根据它们之间的语义相似度进行匹配。元素名本身表示了一定的语义信息，但是元素命名的随意性给语义匹配更是增加了复杂性。在第 3 节，我们提出将参数中的项聚类，同一个簇中的项具有相同或相近的语义，这为元素名匹配提供了依据。另一方面，英语词典 WordNet 也提供了单词之间的语义相似性。本节从这两个方面计算参数名相似度。

WordNet 是由普林斯顿大学的心理学家、语言学家和计算机工程师联合设计的一种基于认知语言学的英语词典[16]。它不但把单词以字母顺序排列，而且按照单词的意义，把所有单词组成一个“单词的网络”。WordNet 最具特色之处，在于它是根据词义而不是词形来组织词汇信息，它已被成功应用于多个自然语言处理系统中[17]。WordNet 支持单词之间的多种连接关系，比如：同义词关系（*synonymy*），*IS-A* 关系（*hypernyms/hyponyms*），*MEMBER-OF* 关系（*holonyms*），*PART-OF* 关系（*meronyms*）。这些关系是用来衡量概念之间距离的重要特征。

式(5)说明了参数名匹配分数的计算方法。其中 $MS_{Concept}$ 表示参数名根据概念簇匹配计算得到的匹配分数，如果两个参数或者组成参数的术语在同一个概念簇中，那么 $MS_{Xoncept}=1$，否则 $MS_{Concept}=0$。$MS_{WordNet}$ 表示根据语义词典 WordNet 计算得到的匹配分数。$\omega_{Concept}$ 为基于概念簇匹配的权重，在本文第 6 节通过实验确定其最优取值。

$$MS_{ElementNane}=MS_{Concept}\times\omega_{Concept}+MS_{WordNet}\times(1-\omega_{Concept}) \quad (5)$$

SemanticMatch 算法（算法 2）基于 WordNet 计算两个元素名称之间的语义匹配分数 $MS_{WordNet}$。如果两个词相同或同义，其相似度分数为最大值 1 或 0.8。如果两个词是层次结构关系（*IS-A*，*MEMBER-OF*，*PART-OF*），那么我们就通过计算它们在 WordNet 空间中的最短路径，来计算两个词之间的语义相似度。两个这样的词之间的相似度分数，等于 0.6 除以两者之间的向量距离。

```
算法 2//基于 WordNet 计算参数名匹配
1  double SemanticMatch (term1, term2)return score{
2  maxScore=1;
3  if(term1 is identical to term2)//两个单词相同
4      score=maxScore;
5  else if (term1 and term2 are synonymous)//两个单词为别名
6      score=0.8* maxScore;
7  else if (term1 and term2 have hierarchical relations)
                  // 两个词是层次结构关系
8      score=(0.6* maxScore)/ N;
                  // N 为层次关系中连接个数
9  else score=0;
10 return score;}
```

除了元素名，每个元素还对应一个值的类型，值的类型是原始数据类型。两个原始数据类型可以是兼容、半兼容或者完全不兼容的。兼容的数据类型包括相同或相似的数据类型，例如 long 和 float。半兼容的数据类型是指彼此在一定程度上相似的数据类型，例如 int 和 float。式(6)说明了值类型匹配分数的计算方法。

$$MS_{ValueType}=\begin{cases}1 & \text{如果两个参数值类型是兼容的}\\ 0.8 & \text{如果两个参数值类型是半兼容的}\\ 0 & \text{如果两个参数值类型不兼容}\end{cases}\quad(6)$$

简单数据类型之间的匹配分数是元素名和元素值类型匹配分数的组合，$MS_{simpleElenent}=\omega_{ElementName}\times MS_{ElemenName}+(1-\omega_{ElementName})\times MS_{ValueType}$。其中：$\omega_{ElementName}$ 取值在 0～1 之间，表示元素名匹配的权重，在本文第 6 节通过实验确定其大小。

4.3 复杂参数类型匹配

复杂数据类型是用户自定义的数据类型，它包含了多个子元素或者属性。我们简单地将属性作为元素来处理，通过匹配其中所有的子元素和属性，计算其平均值作为复杂数据类型匹配分数。在 4.1 的例子中，为了计算两个复杂数据类型 *Item* 和 *ProductParts* 之间的匹配分数，我们构建了一个 2⊗2 的矩阵，如表 1 所示。矩阵中每个单元格的数值，表示相应行和列的两个简单数据类型之间的匹配分数。假设我们得到如表 1 所示的匹配结果，那么 *ProductParts* 中的 *amount* 应当和 *Item* 中的 *quantity* 匹配，匹配分数为 0.8；*ProductName* 应当和 *Product* 匹配，匹配分数为 0.6。如果一个复杂数据类型中又包含另外一个复杂数据类型，匹配过程将递归地进行处理。

表 1 复杂数据类型匹配

Item \ ProductParts	productName:str	amount:int
quantity:int	0	0.8
product:str	0.6	0

复杂数据类型总的匹配分数等于它的每个子元素匹配分数的平均值，如公式(7)所示。根据表 1 中的结果，*Item* 和 *ProductParts* 之间的匹配分数为：$MS=(0.8+0.6)/2=0.7$。

$$MS_{complexElenent}=\frac{\sum_{i=1}^{n}S(subSimpleDatatype)}{n}\quad(7)$$

其中，n 表示源数据类型中简单数据类型和属性的个数。

4.4 接口匹配算法

在介绍了数据类型匹配分数计算方法之后，通过比较源消息和目的消息中的参数列表就可以进行 Web 服务接口匹配。一个消息中封装了多个参数，例如图 3(a)中的 *getDataResponse* 消息包含 3 个部分，*id*、*name* 和 *item*。接口匹配实际上是消息所包含参数的匹配，因此可以通过计算消息所包含参数匹配分数的平均值来计算消息匹配分数。算法 3 说明了计算过程。

在 *matchInterface* 算法中，我们使用一个匹配分数矩阵，来存储所有的源数据类型和目标数据类型对应的匹配分数，算法将识别这两个数据类型列表中所有可能的匹配，并计算这两个数据类型列表之间整个的匹配分数。如算法 3 所述，算法将两列数据类型 *sourceList* 和 *targetList* 作为输入，*sourceList* 包含了 m 个数据类型，*targetList* 包含了 n 个数据类型。根据这两列数据类型，构建一个 $m\otimes n$ 矩阵，矩阵的行对应于源数据类型，矩阵的列对应于目标数据类型。在矩阵的每个单元格中存储对应的行与列的两个数据类型之间的匹配分数。如果对应行与列中的数据类型都是简单数据类型，*matchInterface* 算法将调用简单数据类型匹配算法 *matchSimpleTypes* 来计算匹配分数，并将此分数存入相应的单元格中。如果相对应的数据类型中的一个或两个都是复杂数据类型，那么 *getCompositeDataElements* 将抽取出复杂数据类型中所有的子元素，形成一个新的数据类型列表，进一步进行递归匹配。当矩阵中所有的单元格填满之后，算法就可以考虑所有的参数匹配关系，计算出源数据类型列表各个参数最大的匹配分数的平均值，并返回具有此最大匹配分数的两列参数之间的对应关系。

```
算法 3//计算接口匹配分数
1  procedure matchInterface (sourceList(m), targetList(n))return
   maxScore and paraMapping{
   //输入为源消息和目的消息的两列参数，输出为最大匹配分数
   和参数匹配关系
   matrix=construct a m⊗n matrix;
2  //根据参数列表，构建 m⊗n 匹配分数矩阵
3  //exhaustive matching
4  for (int i=0;i<m; i++){//源消息中有 m 个参数
5      for (int j=0;j<n; j++){//目的消息中有 n 个参数
6          sourceType=sourceList(i)
7          targetType=targetList(j)
8          if (both sourceType and targetType are primitive)
           //简单参数匹配
9  matrix[i,j]=matchSimpleTypes(sourceType, targetType);
10             else if (either sourceType or targetType is complex)
   {//复杂参数匹配
11     newSourceList=getCompositeDataElements(sourceType);
12     newTargetList=getCompositeDataElements(targetType);
       //得到复杂数据类型中的子类型，递归调用匹配算法
13 matrix[i, j]=matchInterface (newBaseList, newTargetList);}}}
14 compute average maximum score and mapping relation among
   parameters;
           //根据匹配分数矩阵，为了得到最大匹配分数，找出参
           数之间对应关系，返回源消息中参数匹配分数平均值
           和源、目的参数之间的对应关系
15 return results;}
```

我们使用 4.1 中的例子来说明该算法。假设我们要匹配图 3(a)中操作 getData 的输出消息 getDataResponse，和图 3

(b)中操作 getProduct 的输出消息 getProductResponse。我们构建一个 3⊗4 矩阵(如表 2 所示)。矩阵中的每个单元格,都填充了一个相应行和列的数据类型之间的匹配分数。假定复杂数据类型 item 和 part 的匹配分数,按照 4.3 节中的算法,计算结果为 0.7,得到如表 2 所示的匹配结果。我们还必须要为源接口中的每个数据类型找到所对应的目标数据类型,源接口中的每个数据类型,在目标接口中只能有一个相匹配的数据类型,这对于目标接口来说也是同样的。为了求得源服务接口的匹配分数最大值,我们需要考虑源接口中各个数据类型和目标接口中各个数据类型之间所有可能的匹配。在这个例子中,两列参数之间的匹配关系一目了然。因此,getDataResponse 和 GetProductResponse 之间的匹配分数为 (0.8+1.0+0.7)/3=0.83。

表 2 Web 服务接口匹配例子

getProductResponse / getDataResponse	id: int	name: str	price: float	part: productParts
id: str	0.8	0	0	0
name: str	0	1.0	0	0.5
items: item	0	0	0	0.7

5 基于接口匹配的 Web 服务自动组合

一个服务是由多个操作组成的,每个操作的输入消息和输出消息,提供了执行操作所需要的参数类型和返回结果的数据类型。例如,一个服务具有一个操作 *buybook*,其输入为 *isdnCode*,另一个操作 *getISDN* 的输出值刚好是 *isdnCode*,因此后一个操作可以同前一个操作进行组合。根据这一思想,我们可以进行 Web 服务的组合。

根据 Web 服务组合的要求,服务请求者会为目的服务提供一个输入信息和目的输出消息。我们用 *inputList* 表示服务请求者提供的输入消息,用 *Goal* 表示服务请求者的目的输出消息,用两个参数列表表示现有基本服务的接口消息:*canInList* 表示现有基本服务中操作的输入消息参数列表,*canOutList* 表示现有基本服务中操作的输出消息参数列表。算法 4 给出了 Web 服务组合的过程。为了方便说明,我们给出了如下定义,其中匹配分数阈值可由用户自行设定。

定义 2 $DT_1(O_1,O_1,\cdots,O_n)$ 和 $DT_2(S_1,S_2,\cdots,S_m)$ 表示两列参数,如果 $\forall O_i \in DT_1$,$\exists S_j \in DT_2$,其匹配分数$(O_i, S_j) \geqslant threshold$,我们称 DT_1 满足$(satisfy)DT_2$。

算法 4//基于接口匹配的 Web 服务组合算法

```
1  serCompose (inputList, goal, canInList (k), canOutList (k))
   return comSerList {
2  for (int i=0; i <k; i++){
   //计算目的输出消息与候选操作输出消息匹配
3  outputMatchScore(i)=matchInterface (goal, canOutList (i))}
4  DescOutList=descendSort (canOutList);
   //对匹配分数结果按降序排列
5  for (i=0; i<l; i++){//l≤k,过虑掉匹配分数较小的候选操作
6     {if (DescOutList (i)satisfies goal)
7        if (inputList satisfies canInList (i))
8           comSerList. add (DescOutList (i));
9           return comSerList;
            //满足消息映射依赖,组合成功,返回结果
10       else//进行输入消息合成依赖匹配
11          unMatched=getLeftParts(canInList (i), inputList);
12          comSerList. add serCompose (inputList, unMatched,
            canInList (k),canOutList (k));//递归调用组合算法
13    else//进行输出消息合成依赖匹配
14       unMatched=getLeftParts(goal, DescOutList (i));
15       comSerList. add serCompose (inputList, unMatched,
            canInList (k),canOutList (k));}}
16 return comSerList;}
```

算法 *serCompose* 首先按照本文第 4 节提出的接口匹配算法,计算 *goal* 和每个候选输出消息(*canOutList*)之间的接口匹配分数,并将这些匹配分数按降序排列。然后,算法核对每个输出消息是否满足 *goal*。若不满足,则对 *goal* 中余下的未匹配的参数部分,进一步做输出消息匹配。这是通过算法本身的递归调用来实现的,递归调用的次数由用户决定。如果输出消息满足 *goal*,则核对用户提供的输入消息参数是否满足该操作的输入消息。如果不满足,那么 *getLeftParts* 过程将在该操作输入消息中,找出未匹配的数据类型,进一步递归组合。在 *goal* 和每个候选输出消息进行匹配后,用户可以使用匹配分数阈值来过滤一些不相关的消息,一些具有较小输出消息匹配分数的操作可以忽略,从而也减轻系统的负担。根据定义 2,用户可以选择不同的阈值进行灵活的服务组合。阈值越大,匹配出的服务的准确度越高,从而组合服务精确度也越高。当然,如果用户减小阈值,降低匹配标准,接口匹配算法能够发现更多的服务,从而提供更多的服务组合机会供用户选择。本文第 6 节通过实验确定了具有最优操作查准率和查全率的接口匹配分数阈值。如果一个输出消息满足 *goal*,并且用户提供的输入消息满足该操作的输入消息,那么这个操作被加入到组合服务列表中。最后,算法返回一个服务操作列表 *comSerList*,也就是满足用户请求的组合服务。

6 接口匹配算法评价

本文 Web 服务组合方法的基本思想,是通过服务接口的匹配发现潜在的 Web 服务,从而实现自动、动态的 Web 服务组合。因此,接口匹配是 Web 服务组合的关键,接口匹配算法的好坏直接影响到服务组合的质量。本节通过实验来评价接口匹配算法,并确定接口匹配算法中未确定的权重等参数。

6.1 实验环境

为了实现纺织企业织厂内部、织厂与上下游企业及合作企业间的高效商业信息交互与集成,我们课题组利用 Web 服务技术,结合企业资源管理的知识以及纺织企业特色,研究和开发了适合纺织企业生产运营的生产管理信息系统(本文以下简称 TRP 系统)[18]。该系统包括 319 个 Web 服务、1207 个操作,我们在该系统的 Web 服务子集上进行实验。实验样本数据包含 392 个操作,在选择样本数据的时候,尽量保证数据分布于系统各个模块中,以保证样本数据的一般性。

6.2 实验结果

我们实验的目的是评价本文提出的接口匹配算法,同时确定接口匹配算法中未确定的参数。参数名匹配是基于聚类算法形成的概念簇和 WordNet 语义词典,所以需要确定基于概念簇匹配的权重。在简单数据类型匹配中结合了参数名匹配和参数值类型匹配,所以需要确定参数名匹配的权重。为了得到最优的查准率和查全率,在接口匹配算法中需要确定接口匹配分数的阈值。我们使用查准率和查全率作为评价接口匹配效果的指标。定义 R 为手工确定的正确匹配,P 为匹

配算法返回的匹配结果，I 是两者的交集，即基于匹配算法得出的正确匹配结果，则

查准率(Precision)：$\Pr ecision=\frac{|I|}{|P|}$

查全率(Recall)：$\mathrm{Re}\ call=\frac{|I|}{|R|}$

为了确定基于聚类算法形成的概念簇匹配在参数名匹配中的权重 $\omega_{Concept}$，我们设定参数名匹配的权重 $\omega_{ElementName}$ 为 0.8，并在 0.1～1 之间改变 $\omega_{Concept}$ 的值，得到如图 4 所示的查准率、查全率结果。从实验结果可以看出，查准率和查全率随着 $\omega_{Concept}$ 的增大先增加后减小，$\omega_{Concept}=0.56$ 是查准率变化的转折点，$\omega_{Concept}=0.6$ 是查全率变化的转折点。因此，我们选择 $\omega_{Concept}=0.58$ 作为参数名匹配中基于概念簇匹配的权重大小，以取得最佳查准率和最佳查全率的折衷。从图 4 可以看出，在 $\omega_{Concept}$ 逐渐增大的初期，查全率的变化比查准率变化要大，这是因为概念簇中的项并不能完全代表整个参数的语义，因此降低了匹配结果的精度；而对于正确拼写的英语单词，WordNet 能够准确计算单词之间的语义，从而提高了查准率。这也正是查准率、查全率经过 $\omega_{Concept}$ 转折点后，查准率减小的速度要小于查全率减小速度的原因。

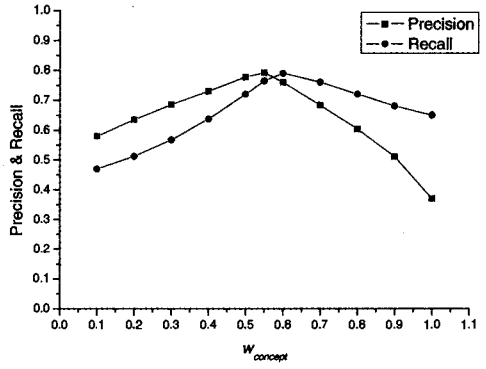

图 4　查准率和查全率随权重 $\omega_{Concept}$ 变化实验结果

为了确定简单类型匹配中基于参数名匹配的权重 $\omega_{ElementName}$，我们设定概念簇匹配的权重 $\omega_{Concept}$ 为 0.58，并在 0.1～1 之间改变 $\omega_{ElementName}$ 的值，得到如图 5 所示的查准率、查全率结果。从结果中可以看出，$\omega_{ElemenName}=0.87$ 是查准率和查全率变化的转折点，因此我们确定参数名匹配的权重为 0.87。从图 5 可以看出，参数名匹配起了主要的作用，而只有在参数命名随意性很大的情况下，参数值类型匹配才能为参数匹配起到一定的作用。因为本节的实验样本数据取自于同一个应用系统中，其参数命名采用统一的规则，这使得参数值类型匹配的作用甚小。在异构程度很高的企业合作伙伴服务接口匹配中，参数值类型匹配会起到相对较大一些的作用。在 $\omega_{ElementName}=0.87$ 转折点之前，查准率随 $\omega_{ElementName}$ 的变化快速增加，这是因为参数的语义主要体现在参数名部分，而查全率随 $\omega_{ElementName}$ 的变化缓慢减小，这部分减小是因为不规范命名的部分参数引起的。实验结果同时说明：参数名匹配能够克服参数值类型匹配的弱语义性，而参数值类型匹配能够克服参数名匹配对不规范命名参数的无效性，二者的结合使得参数匹配具有最优的性能。

在基于接口匹配的 Web 服务组合算法中，接口匹配分数阈值的选择，使得用户可以进行精确或者松散的服务组合。为了实现在一定程度上既精确、又不失去可能组合机会的 Web 服务组合，我们通过改变接口匹配分数的阈值，计算查准率和查全率，以确定具有最优查准率和最优查全率折衷的接口匹配分数阈值。在 0.1～1 之间改变阈值，得到如图 6 所示的实验结果。从结果可以看出，查准率随着阈值的增大逐渐增大，查全率随阈值的增大逐渐减小，二者相交位置的接口匹配分数为 0.76，由此我们确定接口匹配分数的阈值为 0.76。如果用户希望更精确的 Web 服务组合，那么可以通过增大阈值，反之，如果希望得到尽可能多的操作组合可能，那么用户可以减小接口匹配分数阈值。

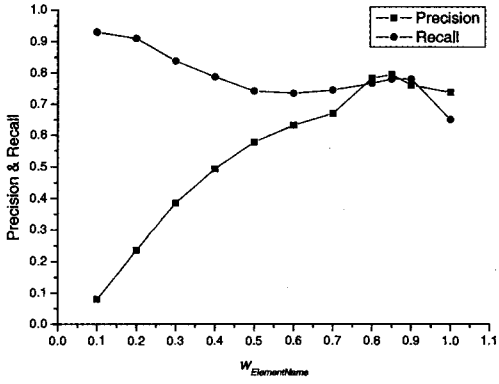

图 5　查准率和查全率随权重 $\omega_{ElementName}$ 变化实验结果

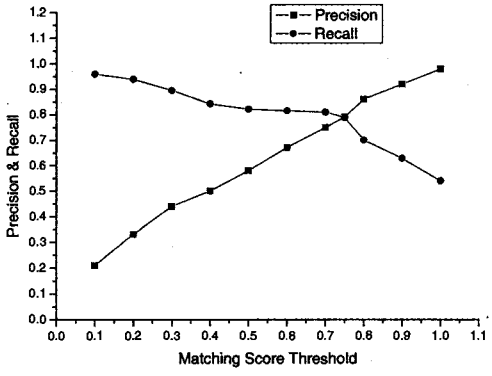

图 6　接口匹配分数阈值的确定

为了验证参数聚类和 WordNet 语义词典对接口匹配性能的影响，我们对以下四种情况进行了实验：(a)没有使用概念簇匹配和 WordNet；(b)只使用概念簇进行匹配；(c)只使用 WordNet 进行匹配；(d)结合概念簇匹配和 WordNet 语义词典匹配，我们得到如图 6 所示的实验结果。可以看出方法(a)的查准率不足 0.1，这说明直接根据参数进行匹配不具有实际意义。虽然查全率略高一些，但也不超过 0.2，其略高的原因是因为参数值类型匹配在一定程度上提高了查全率。方法(b)和(c)使得查准率和查全率至少有 40%的提高，这说明参数聚类和 WordNet 语义词典确实能提高接口匹配的性能。WordNet 匹配相对于概念簇匹配更能改善查准率的性能，这是因为 WordNet 对于正确拼写的英语单词，能够准确地计算其语义匹配度。而概念簇匹配比 WordNet 更能显著提高查全率的性能，这是因为聚类算法通过对接口参数的聚类，克服了参数命名随意性的缺陷，对非正确英语拼写的参数特别有效。由此可见，概念簇匹配和 WordNet 匹配互为补充，二者的结合既能有效计算正确拼写参数名的语义近似程度，又能克服异构环境下 Web 服务接口定义的随意性。从图 6 的实验结果也可以看出，结合概念簇和 WordNet 的匹配方法(d)相对于仅仅使用其中之一的方法(b)或(c)，对查准率和查全

率均有 20%～30%的提高。

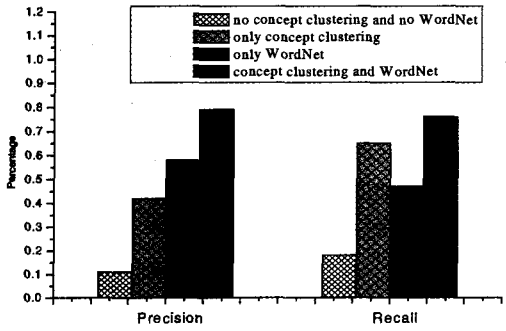

图 6　概念簇和 WordNet 对接口匹配性能的影响

结论　通过把服务抽象成为具有输入和输出接口的实体，本文提出了基于接口匹配的 Web 服务组合方法。它能够实现在现有服务描述标准基础上自动、动态发现潜在的 Web 服务交互。WSDL 是 Web 服务接口描述的标准语言，得到工业界的广泛支持。因此，我们利用 WSDL 文档已经提供的信息，进行服务接口匹配。接口匹配算法基于消息参数聚类形成的概念簇和现有的 WordNet 语义词典。实验结果证明二者的结合互为补充，从而使得匹配算法具有更优的查准率和查全率。在此基础上提出了 Web 服务组合的算法。

WSDL 对接口的描述只限于语法层次，不能表达语义信息。本文的接口匹配算法借助于参数聚类和 WordNet 语义词典，在一定程度上克服了这一缺陷。如果各个服务的描述都能够遵循统一的语义约定，在某个应用环境中共同遵守的语义词汇集合范围内描述 Web 服务，服务之间将能相互识别，实现自动的组合和协作。因此，在今后的工作中，我们将进一步研究基于语义的 Web 服务组合。

参 考 文 献

1　Benatallah B, Dumas M, Fauvet M, et al. Towards Patterns of Web Services Composition [M]. In: Patterns and Skeletons for Parallel and Distributed Computing, Springer Verlag, UK, 2003. 265～296

2　Rao J H, Su X M. A Survey of Automated Web Service Composition Methods [A]. In: Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004), San Diego, USA, July 2004. 43～54

3　Srivastava B, Koehler J. Web Service Composition - Current Solutions and Open Problems [A]. In: ICAPS 2003 Workshop on Planning for Web Services, Trento, Italy, June 2003. 51～59

4　Milanovic N, Malek M. Current Solutions for Web Service Composition [J]. IEEE Internet Computing, 2004, 8(6): 51～59

5　Casati F, Shan M C. Definition, Execution, Analysis, and Optimization of Composite E-Services [J]. IEEE Data Engineering Bulletin, 2001, 24(1): 29～34

6　Sheng Q Z, Benatallah B, Dumas M, et al. SELF-SERV: A Platform for Rapid Composition of Web Services in a Peer-to-Peer Environment[A]. In: Proceedings of 28th Very Large Data Bases, Hong Kong, China, August 2002. 1051～1054

7　The OWL-S Service Coallition. OWL-S: Semantic Markup for Web Services, version 0. 1. http://www. daml. org/services/owl-s/1. 0/owl-s. pdf

8　McIlraith S, Son T, Zeng H. Semantic Web Services [J]. IEEE Intelligent Systems, 2001, 16(2): 46～53

9　Christensen E, Curbera F, Meredith G, et al. Web Services Description Language (WSDL) 1. 1. http://www. w3. org/TR/wsdl, March 2001

10　Casati F, Shan M C. Models and Languages for Describing and Discovering E-Services (Tutorial) [A]. In: Proceedings of the International ACM SIGMOD Conference on Management of Data, Santa Barbara, California, USA, May 2001

11　Hand D, Mannila H, Smyth P. Principles of Data Mining[M]. Cambridge, MA, USA: The MIT Press, 2001

12　Kaufman L, Rousseeuw P J. Finding Groups in Data: An Introduction to Cluster Analysis [M]. New York: John Wiley & Sons, 1990

13　Zaremski M, Wing J M. Signature Matching: a Tool for Using Software Libraries[J]. ACM Transactions on Software Engineering and Methodology, 1995, 4(2): 146～170

14　Zaremski M, Wing J M. Specification Matching of Software Components[J]. ACM Transactions on Software Engineering and Methodology, 1997, 6(4): 333～369

15　Luckham D C, Vera J, Meldal S. Three Concepts of System Architecture[R]: [Technical Report. CSL-TR-95-674]. Stanford University, 1995

16　Miller G. WordNet: An On-line Lexical Database[J]. International Journal of Lexicography, 1990, 3(4): 235～312

17　Voorhees E M. Using WordNet for Text Retrieval. In: Fellbaum C, ed. WordNet: An Electronic and Lexical Database[M]. Cambridge, MA, USA: The MIT Press, 1998. 285～303

18　Yu S J, Le J J. Study and Development of Textile Enterprise Management System Based on Web. In: Proceeding of 8th Joint International Computer Conference (JICC), Ningbo, China, November 2002. 121～125

(上接第 53 页)

架和相应的三方安全协议，除了满足大规模存储系统的可扩展性要求，还保证该存储系统的系统级安全性。

借助形式化分析方法，对三方安全协议进行了逻辑推导，从推导结果可知：在一定的假设下，三方安全协议能保证请求中的权能标识的完整性、协议传输过程的正确性，以及消息的真实性，从逻辑上验证了三方安全协议的正确性和可行性。

网络技术和存储技术不断发展，网络存储面临新的安全挑战，比如新的攻击方式。为此，我们将在进一步工作中修改权能标识结构，并用形式化分析方法指导三方安全协议的详细设计。

参 考 文 献

1　Singh A, Voruganti K, Gopisetty S, et al. Security vs Performance: Tradeoffs Using a Trust Framework. In: Proceedings of the 22nd IEEE/13th NASA Goddard Conference on MSST, 2005

2　Neumann C, Ts'o T. Kerberos: An Authentication Service for Computer Networks. IEEE Communications Magazine, 1994, 32 (9): 33～38

3　Blaze M. A Cryptographic File System for UNIX. In: Proc. of 1st ACM Conference on Communications and Computing Security. USA: ACM Press, 1993. 9～16

4　Gibson G A, Nagle D F, Amiri K, et al. File Server Scaling with Network Attached Secure Disks. In: Proc. of the ACM ICMMCS. USA: ACM Press, 1997. 272～284

5　Kubiatowicz J, Bindel D, Chen Y, et al. Oceanstore: an Architecture for Global-Scale Persistent Storage. In: ASPLOS-1x. USA: ACM Press, 1999. 190～201

6　Butler M L. 1 Petabyte Production Storage Environments and File Systems. In: 2004 International Conference on Supercomputing. USA: ACM Press, 2004

7　Dennis J B, Van Horn E C. Programming Semantics for Multiprogrammed Computations. Communications of the ACM, Feb. 1966

8　卿斯汉. 安全协议 20 年研究进展. 软件学报, 2003, 14(10): 1740～1752

9　Gong L, Needham R, Yahalom R. Reasoning about belief in cryptographic protocols. In: Proc. of the 1990 IEEE Computer Society Symposium on Research in Security and Privacy. USA: IEEE Press, 1990. 234～248

10　Fabrega F J T, Hertzog J, Guttman J. Strand spaces: Proving security protocols correct. Journal of Computer Security, 1999, 7 (2-3): 191～230