维普资讯 http://www.cqvip.com

基于软件能力模型的匹配机制*)

于玮萍 王 茜
(东南大学计算机科学与工程系 南京 210096)

摘 要 国际标准ISO 16100制定了软件能力描述的规范,提出了加强软件互操作和数据交换能力的方法学。基于该国际标准,本文研究了软件互操作和数据互交换的实现机制;分析了在不同情况下软件匹配的规则,尤其针对遗留系统的互操作,分析了各种进行语义匹配的方法,并提出了相应的系列算法;原型系统的实现,验证了算法的可行性、合理性;通过对算法的时间复杂度的分析,说明了算法的实用性。

关键词 互操作,能力匹配,语义匹配,对应节点,相关节点

Matching Mechanisms Based on Software Functionality Models

YU Wei-Ping WANG Qian
(Department of Computer Science and Engineering,Southeast University,Nanjing 210096)

Abstract The international standard ISO16100 specifies the manufacturing software capability profiling for interoperability and puts forward the methodology which could enhance the interoperation between software and the ability of data exchanges. The mechanisms which could make these interoperation and the data exchanges to be implemented are studied in this paper. A set of matching rules under different circumstances and various methods of semantic matching for interoperability,especially when legacy systems are under consideration are discussed. The feasibility and rationality of the corresponding algorithms have been tested in the implementations of the prototype system. The practicability is demonstrated through the analysis for Time Complexity of the algorithms.

Keywords Interoperation,Capability matching,Semantic matching,Corresponding nodes,Relating nodes

1 引言

理想状态下,电子数据的交换、管理和访问都是流畅和无缝的。用户只需要在信息系统中输入一次必要的信息,这些信息就能通过网络传递给所有需要的人或应用,也就是说企业软件可以使业务过程自动化。这样的自动化是由过程中各个活动的协同来保证的,协同之一意味着完成各个活动的软件系统能够进行数据交换、实现互操作。这样的互操作就要求提高协议的标准化程度。

国际标准ISO 16100就是通过对软件能力进行规范化的描述来提高数据交换、软件互操作的能力的。这种描述是可计算机表达、人可读取的。它的目标是提供描述制造软件能力的方法,这些能力与软件在其整个制造应用的生命周期中的任务相关,不依赖于具体系统体系结构和执行平台。换句话说,它提供一种能力建模的方法,能够独立地表述制造软件在整个生命周期中的能力。由于这种建模的系统定义独立地存在于任何实现模型之外,又能包含行业细节制定的规范[1],因此可以提高系统之间的互操作水平。

本文首先介绍这种软件能力建模的方法,进而分析和研究基于这种描述方法实现软件互操作、进行需求匹配的机制。

2 软件能力建模

软件能力建模过程中有4个基本元素:活动树、能力类、能力模板和能力描述。活动树是领域中所有“活动”的树型结构,定义了一个制造应用的领域,由活动节点组成。每个节点对应一个能力类,表明软件所应该提供的“能力”,其中包括软件提供的功能、所需的资源和软件所处理的信息。每个能力类都可以由能力模板来具体表述能力描述的格式。针对某个具体的软件,通过填写能力模板,即赋予能力模板中每个元素以具体值,得到一个软件的能力描述(Profile)。这样的描述是规范的、符合标准的。

当用户提出一个需求、按照需求搜索/寻找相应的制造软件时,首先需要对用户需求的能力进行描述REQ(REQuirement profile);然后,按照REQ,搜索制造软件的能力描述MSU(Manufacturing Software Unit profile)。也就是说,通过MSU和REQ进行匹配,使制造软件和用户需求进行匹配。

能力模板定义了能力描述的格式[1],它由两部分组成:共有部分和特有部分。共有部分主要描述能力的基本属性(例如属于的应用领域、活动、REQ或是MSU等)、软件运行的环境(即条件约束,例如操作系统、计算机系统等)。因此,对于共有部分的匹配涉及到对诸如硬盘空间大小、使用系统是否兼容等的条件判断、域值比较等,这在文[2]中已有详细的论述。本文主要探讨特有部分的匹配。特有部分中主要描述的是完成该能力的活动以及伴随该活动的各种输入和输出信息[1]。

3 能力描述的简单匹配

3.1 问题描述

上节中提到的能力建模中4元素之一的“活动树”,以树型结构的形式定义了对应的应用领域中的活动的组织,又名

*)国家“八六三”计划基金资助项目(2003AA414110)。于玮萍 硕士研究生;王 茜 博士生导师。

维普资讯 http://www.cqvip.com

参考的能力类结构 RCCS(Reference Capability Class Structure)。一个节点表示一个活动,一个活动由一个功能模块来完成;子节点表示子活动,由子功能模块来完成,见图 1 RCCS。RCCS 中的节点标识由层次-名称- ID 组成,例如 L1:DevelopProducts(AA),L1 表示是该节点处于第 1 层,其中 L 表示层次(Level);DevelopProducts 是该功能节点的名称(Name);(AA)中 AA 是该节点的 ID一节点的唯一标识,同时表示了该节点在 RCCS 中的位置,即指示了从根节点到该功能节点的路径,简称为路径标识。方便起见,下面提及节点(xx),意指此 ID 为 xx 的节点;而对于节点 xx,意指名称为 xx 的节点。

能力建模的方法学规定:能力描述的特有部分必须选自所属的应用领域的某个 RCCS,即是 RCCS 节点的一个子集。简单匹配指参加匹配的 msu 和 req(msu 指 MSU 的特有部分,req 指 REQ 的特有部分)选自同一 RCCS,如图 1 所示。另外,能力描述的特有部分的节点集合是指能够表述所描述的软件所需要(或完成)功能的个数最少的节点集合(或者说,能够覆盖所描述的软件所需要(或完成)功能的最高层的节点集合)。例如,如果某 req 含有 a 节点,则表示 req 需要节点 a 及其所有子(孙)功能;如果 req 不包含 a 的所有后代节点的功能,那么在生成 req 时,只能在 RCCS 上选取 a 中部分的、所需要的子功能节点,而不能直接选取 a。图 1 中 req 选择了 RCCS 中的 L6:ScheduleInspection(AA22131)、L4:GenerateOperations(AA222)、…等,表示所需要的软件应该具有这些功能节点所表述的功能;msu 选择了 RCCS 中的 L5:ScheduleHandling(AA2212)、L5:ScheduleProcessing(AA2213)、…等,其中(AA2213)含有子功能节点(AA22131)和(AA22132)。而(AA22132)还含有子功能节点,表示 msu 能够完成这些功能节点所表述的功能及其所有的子功能。

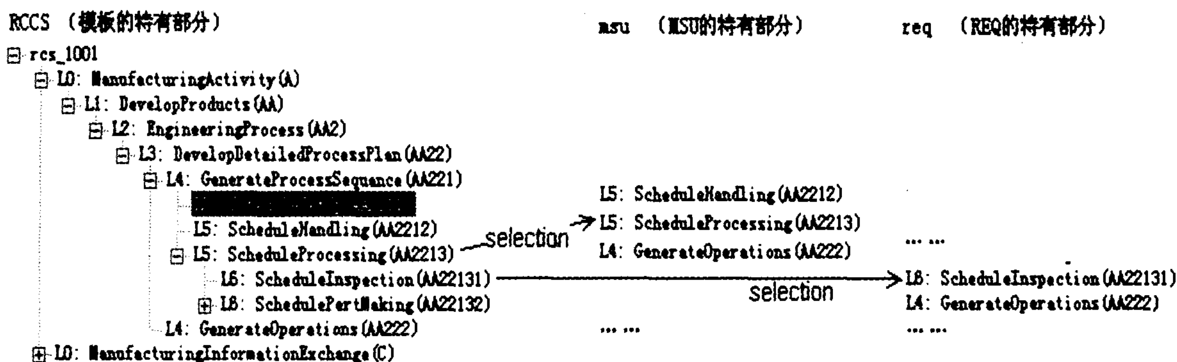


图 1 能力模板中的特有部分

若 msu 所含功能集覆盖了 req 的需求功能集,匹配成功,即找到与需求相符的软件单元。否则,就会出现部分匹配或不匹配的情况。如果部分匹配或不匹配,就需要有详细的列表来说明 req 的每项需求的匹配情况,哪些节点匹配,哪些节点不匹配。如果匹配,匹配到什么程度,以便根据这些情况来考虑是调整需求、重新匹配,还是开发新的软件以满足需求。所以,需要对 req 中每个节点给出匹配级别,称之为单元匹配级别,它可以分为 3 级:

对于 req 中节点 a:

①精确匹配(E)。若 msu 上存在节点 b,b 包含了 a 的功能,则称对 a 精确匹配;

②弱匹配(W)。若 msu 上存在节点 b,b 只实现了 a 的子功能,则称对 a 弱匹配*;

③不匹配(N)。若找不到对 a 精确匹配或弱匹配的节点 b,则称对 a 不匹配。

在 msu 中搜索 req 的过程称之为单元匹配(Unit Matching)。

在得到 req 所有节点的单元匹配级别后,获得综合情况,根据一定的原则(见 3.2 和 3.3),给出 req 与 msu 匹配的结果。

3.2 简单匹配的算法

定义 1 当 req 和 msu 选自同一 RCCS 时(简单匹配),若 req 中的一个节点和 msu 中的一个节点在 RCCS 上有祖孙(父子)关系,称 msu 中该节点是 req 中该节点的对应节点,称该对节点有对应关系。

根据对应节点的位置可以得到简单匹配中单元匹配级别的判断:

a)对应节点是 req 中该节点的祖先或其本身,这意味着 msu 完成的功能覆盖了 req 的需求,按单元匹配级别定义,为精确匹配 E;

b)对应节点是 req 中该节点的后代节点,这意味着 msu 只完成了该需求节点的子功能,为弱匹配 W;

c)没有找到对应节点,即不匹配 N。

从图 1 及 ID 的定义不难得到:当节点 a. ID 为 b. ID 的前缀时(比如 a. ID=“A”,b. ID=“AB”),a 在 RCCS 上处于 b 的祖先位置(注:我们提到前缀时不包括两字符串相等的情况)。由此得:

算法①:简单匹配算法 1

输入:req 和 msu

输出:匹配是否成功及相应匹配列表

1)bool success=true(默认匹配成功);

2)遍历 req 中每个节点:

对于 req 的每个节点 a,遍历 msu 的每个节点:

a)如果存在节点 b,使得 b. ID 是 a. ID 的前缀或其本身,则在匹配列表中添加表项,指示对 a 找到精确匹配;

b)如果存在节点 b,使得 a. ID 是 b. ID 的前缀,则在匹配列表中添加表项,指示对 a 找到弱匹配,success=false;

* 一旦出现弱匹配,可以提示:将 req 的此节点进行分解(依据 RCCS),以得到更细化的单元,重新匹配;根据重新匹配的结果来做调整和优化。

维普资讯 http://www.cqvip.com

c)否则,在匹配列表中添加表项,指示对 a 没有找到匹配项,success=false。

3)若 success 为 true,匹配成功;否则,匹配失败。

即当且仅当对所有 req 节点找到精确匹配时认为匹配成功。

3.3 匹配级别问题

算法①表明,只有精确匹配才能满足匹配的要求。如果对 req 中每个节点都这样要求是非常苛刻的,在现实中没有必要。为了使得匹配更加细化、灵活和实用,应该明确标注出必须进行匹配的元素,也就是说,要求对 req 中每个节点都注明"可选"还是"必须"。这样,在进行匹配时,对"可选"节点不必介意是否匹配;而对于"必须"的节点则要求单元匹配级别为精确匹配。这样就减少了需要精确匹配的节点。根据单元匹配的结果汇总,可以得到简单匹配的级别,分为 4 类:完全匹配;不匹配;所有必须条件匹配;一些必须条件匹配,如表 1 所示。

表 1 简单匹配的级别

Req 中所有"必须"节点的单元匹配级别	Req 中所有"可选"节点的单元匹配级别	req 和 msu 的匹配级别
仅含无匹配或弱匹配项(N|W)	——(不做参考意见)	不匹配
至少有一个节点(但非全部)达到 E	——	一些必须条件匹配
均达到精确匹配 E	至少有一个节点没有达到 E	所有必须条件匹配
均达到精确匹配 E	均达到精确匹配 E	完全匹配

考虑匹配级别,得算法①′:简单匹配算法 2(若 req 无"必须"节点,则匹配无意义,所以 req 的"必须"节点个数>0)

输入:req 和 msu

输出:匹配级别

1)Int MandatoryCount=0; //记录 req 中标识为"必须"的节点个数

Int MandatoryE=0; //记录 req 中精确匹配的标识为"必须"的节点的个数

Int OptionalCount=0; //记录 req 中标识为"可选"的节点个数

Int OptionalE=0; //记录 req 中精确匹配的标识为"可选"的节点的个数

2)遍历 req 中每个节点:

a)对于 req 的每个节点 a,遍历 msu 的每个节点。若 a 为"必须":MandatoryCount++;若 a 找到精确匹配,MandatoryE++;

b)若 a 为"可选":OptionalCount++;若 a 找到精确匹配,OptionalE++;

3)if(MandatoryE==0) 输出:不匹配

```
else{
  if(MandatoryCount != MandatoryE)
    输出:一些必须条件匹配
  else if(OptionalCount == OptionalE)
        输出:完全匹配
    else 输出:所有必须条件匹配
}
```

算法结束。

4 语义匹配问题

4.1 问题描述

能力建模中的第一要素就是 RCCS,在简单匹配中它起着关键性的作用。而 RCCS 的定义相当困难,它是行业的标准,必须由行业的专家统一观念,并经过反复的讨论来制定,这是未来的理想状态。目前大量的软件系统,我们称之为遗留系统的,都还是建立在不同的 RCCS 之上的。同一领域,遗留系统所采用的 RCCS 的定义不同。这些相同领域的、不同的 RCCS 具有不同的树形结构、不同的深度、不同的高度;同一个活动可能有不同的名称、不同的 ID(反映了该活动具有不同的路径、处在树形结构的不同的位置)。那么,简单匹配就不能发挥作用,必须考虑如何对遗留系统进行 req 和 msu 的匹配。

我们将 req 选自的 RCCS 称为源 RCCS,将 msu 选自的 RCCS 称为目标 RCCS。同样的功能节点在源 RCCS 上的 ID 为 A,在目标 RCCS 上 ID 为 B,对于含节点(A)的 req 和含节点(B)的 msu(说明:为简化问题,msu 不含节点(A)),按照功能匹配来说,此 msu 可以满足该 req 对节点(A)功能的需求。但按算法①(按 ID 匹配),req 的需求找不到匹配项。因此我们需要更高级的匹配算法,它们能找出目标 RCCS(B)与源 RCCS(A)的对应关系,且给出正确的匹配结果。本文把它称为语义匹配。

4.2 语义匹配的算法研究

4.2.1 语义匹配的分析

由 4.1 的分析可知,语义匹配最重要的就是找出相同功能节点的对应关系。

假设 1 如果 req 的节点名称(Name)和 msu 的节点名称同义(即在字典中它们属于同义词)或相同,我们就认为此对节点为同功能节点。

定义 2 对于一个 RCCS 上的节点 a,若在另一个 RCCS 上查找到节点 b,使 a. Name 和 b. Name 相同或字典同义,则称 a 和 b 为相关节点。

推论 1 基于假设 1,相关节点即为同功能节点。由此得:

算法② 语义匹配算法 1

输入:req 和 msu

输出:匹配是否成功及相应匹配列表

1)bool success=true; string [] sameWords(用来存放同义词的字符串数组)清空;

2)遍历 req 中每个节点:

对于 req 的每个节点 a:

a)清空 sameWords;

b) 搜索字典,如果字典含 a. Name 项,将字典中其同义词组(包括它本身)加入 sameWords 中,否则只将 a. Name 放入 sameWords 中。

c)遍历 msu 的每个节点,如果存在节点 b,使 sameWords 中包含 b. Name(即两节点名称同义或相同),则在匹配列表中添加表项,指示对 a 找到匹配;否则在匹配列表中添加表项,指示对 a 无法找到匹配,success=false;

3)若 success 为 true,匹配成功,否则匹配失败。

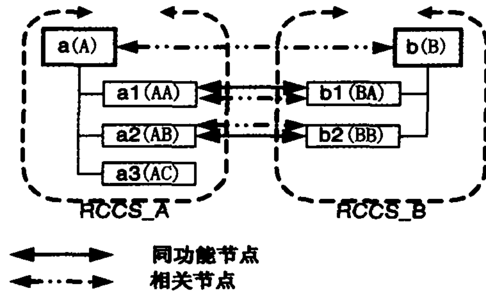
算法②的概念清晰、简单,但有很大的局限性。

(1)算法②依赖于假设 1,但两个 RCCS 中的相关节点的

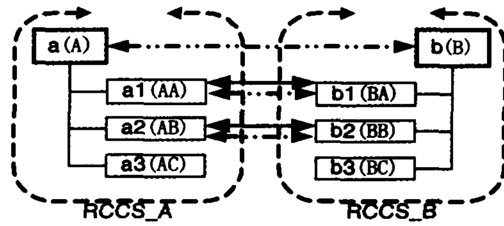
维普资讯 http://www.cqvip.com

子功能定义极可能不同，如图2所示。其中，a和b、a1和b1、a2和b2是相关节点，而只有a1和b1、a2和b2是功能相同的节点，a和b不是功能相同的节点，即推论1不成立。

为此，需要对同功能节点进行进一步的分析。



(1) 子节点有包含关系



(2) 子节点无包含关系

图2 节点功能定义不同

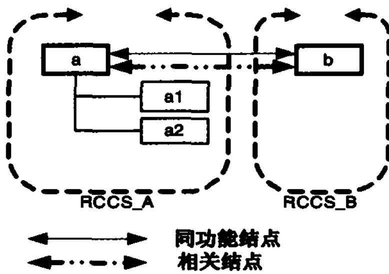


图3 相关节点有一为叶节点

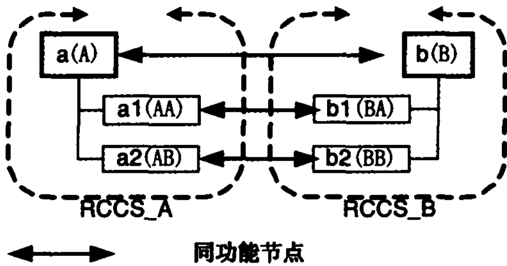


图4 节点功能定义相同

2)若相关节点a和b均含有子节点，对它们含有子节点的情况进行如下分析：

a)子节点无对应，即a和b两节点的功能完全不一样，则认为它们为非同功能节点；

b)子功能节点的划分完全相同，即所有子节点间均一一对应为同功能节点(图4)。说明两节点的定义一样，显然是同功能节点；

c)子节点只有部分对应，任意一种情况，均不认为此相关节点为同功能节点。

①功能包含关系：$a_n \subset b_n | b_n \subset a_n$(其中 a_n 和 b_n 分别指 a 和 b 的子节点集合(图2(1))；

②非功能包含关系：$a_n \cap b_n \neq \Phi \wedge a_n \cup b_n \neq a_n | b_n \wedge a_n \cap b_n \neq a_n | b_n$(图2(2))。

由以上分析得同功能节点的更准确的定义：

定义3 一个RCCS上的节点a和另一个RCCS上的节点b是同功能节点，当且仅当：

1)a和b是相关节点，且

2)a和b至少有一个为叶节点，或者

1)相关节点中至少有一个没有定义子节点(即为叶节点)，如图3中的b节点。由于a和b是相关节点，根据推论1，认为它们功能相同，即b隐式包含a的两个子功能(可以认为不同RCCS的分解粒度不同)，所以a与b为同功能节点。

3)a和b子功能节点的划分完全相同，即所有子节点间均可一一对应为同功能节点。

由3)知，这是一个递归定义。

(2)若req选自图3的RCCS_A，含a；msu选自图3的RCCS_B，含b；a和b是相关节点，按算法②，匹配成功；但若req选自图3的RCCS_A，含a2(a的子节点)；msu仍注册于RCCS_B，含b。msu实际上是覆盖了req的功能需求。但是按照算法②，a2. Name找不到同义词b2. Name，而给出匹配失败的结果。这样就会产生错误的结果，即误匹配(未能发现精确匹配)。

(3)算法②没有弱匹配的结果，即所给出的匹配列表没有提示将需求进行分解的能力。

由以上分析得到语义匹配的单元匹配要求：

① 按照定义3，寻找功能匹配节点；

② 解决误匹配的问题；

③ 区分精确匹配和弱匹配的问题。

4.2.2 语义匹配算法

依据4.2.1分析的语义匹配要求，我们给出两种匹配算法：第一种是req和msu上节点直接进行单元的语义匹配，匹配时分别参考各自的RCCS；另一种是先解决RCCS间的语义映射问题，再将原req进行同义转换至msu选自的RCCS，形成新req。将新req与msu进行简单匹配，得到匹配结果。

4.2.2.1 基于各自的RCCS，req节点和msu节点直接匹配

假设2 如果两个RCCS中的节点是相关节点，则它们的子节点的定义一定能满足同功能节点的定义对应同功能，即它们一定是同功能节点。

推论2 两个RCCS中的相关节点一定能够满足定义3的要求。

由此假设，改进算法②，对req中的节点找到所有精确匹配：

算法③：语义匹配算法2

输入：req和msu

输出：匹配是否成功及相应匹配列表

1)bool success=true;

2)遍历req上每个节点：

对于req每个节点a，遍历msu的每个节点：

a)若存在节点b，使a. Name与b. Name同义，则在匹配

维普资讯 http://www.cqvip.com

列表中添加表项，指示对 a 找到(精确)匹配；

b) 否则，在 a. ID 上截取每个可能的前缀(即其祖先的 ID)，在源 RCCS 上查找它所对应的节点的名称 c；对每个名称 c，遍历 msu 的每个节点：若存在节点 d，使 c 与 d. Name 同义，则在匹配列表中添加表项，指示对 a 找到(精确)匹配 *；否则，在匹配列表中添加表项，指示对 a 无法找到匹配，success=false；

3)若 success 为 true，匹配成功；否则，匹配失败。

该算法由于必须在 a. ID 上截取每个可能的前缀，在源 RCCS 上查找其祖先 c，然后按照 c 遍历 msu，进行匹配，所以时间复杂度显然大于算法②。

该算法查找了 req 的祖先节点的名称，所以解决了误匹配的情况。

为了要找到弱匹配，改进算法③，得：

算法④：在算法③ * 处添加

*对每个 msu 节点 b，截取 b. ID 的每个可能的前缀，在目标 RCCS 上查找它所对应的节点名称，对每个名称查看是否与节点 a 同义：如果是，则在匹配列表中添加表项，表示对 a 找到弱匹配，success=false；

该算法由于必须在 b. ID 上截取每个可能的前缀，在目标 RCCS 上依次查找其祖先，进行匹配，所以显然时间复杂度大于算法③，详见 4.3 分析。

因为算法③和算法④都需要根据 req 或 msu 节点查找其在 RCCS 上相应节点的祖先节点然后再进行遍历和匹配的判断，所以虽然解决了误匹配和弱匹配问题，但是算法效率是非常低的，且它们的匹配是基于假设 2 的。

4.2.2.2 基于 RCCS 间语义映射的匹配

算法③和算法④在进行单元匹配时是基于 req 和 msu 各自的 RCCS 的。换言之，它要在能力描述的节点与 RCCS 之间进行查找，然后再反复地进行遍历，这是导致它低效率的主要原因。以下语义匹配的机制先解决 RCCS 之间的语义映射(即同功能节点映射)，将问题转换为基于相同的 RCCS 进行简单匹配，以此提高匹配效率。

语义映射解决 RCCS 间差异问题(即 RCCS 间的同功能节点查找、映射)，形成映射表并注册入数据库，如表 2 所示。表 2 表示 RCCS_A 节点和 RCCS_B 节点之间映射在 Http://……. AToBmap. xml 的文件中。当建模中 req 和 msu 选自不同的 RCCS 时，我们将 req 上节点按照映射表转换得新的 req，然后用新的 req 和原 msu 进行简单匹配，得到算法⑤。

表 2 RCCSs 之间功能节点映射的索引表

RCCSID1	RCCSID2	Map Uri
RCCS_A	RCCS_B	Http://……. AToBmap. xml
…	…	…

算法⑤：基于语义映射的匹配算法

输入：req 和 msu。已知 req 选自 RCCS_A，msu 选自 RCCS_B。

输出：匹配是否成功及相应匹配列表

1)若 RCCS_A ! = RCCS_B，则执行 2)，否则用算法①进行匹配，得匹配结果，算法结束。

2)查数据库，RCCS_A 与 RCCS_B 是否存在语义映射(表 2)? 若存在，按 uri 取得映射表 M；若不存在，新建映射表 M，且将其注册入数据库。建立新的能力描述-nreq。

3)遍历 req 的每个节点。

对于 req 每个节点 a：

a)若能在 M 中查找到 a. ID 的对应项，则将 a. ID 在 M 中的对应项 nID 填写入 nreq，并在 a 节点后添加信息："->("+nid+")"(表示原节点到转换节点的映射，因为此算法得出的匹配列表项使用的 ID 号是 nreq 的 ID 号，匹配结束后看匹配列表需要此信息来对应，以便得知对原节点 a 的匹配结果)；

b)若在 M 对应栏中查找不到 a. ID，则匹配失败，算法结束。

4)将 nreq 和 msu 作为输入，按算法①(或①')进行匹配，得到匹配结果，算法结束。

第 2)步中新建映射表有两种方式：人工映射和自动语义映射。人工映射即通过人机界面进行简单的连接操作，指出 RCCS_A 中各节点在 RCCS_B 中对应的节点位置。只需连接差异部分，未做连接部分认为一一对应。然后将连接的信息按对存储为映射表，存入数据库。

自动语义映射必须对 RCCS_A 中的每个节点根据字典中的同义词逐个在 RCCS_B 中进行遍历，查找对应节点。

假设 3：RCCS 自身不会出现可语义映射的节点(即每个节点的功能是唯一的)。

如能在 RCCS_B 中找到可映射项(根据同功能节点的定义)，进行映射连接，则将映射项记录入映射表，存入数据库。

由于语义映射是对称的，对于两棵 RCCS，可任选其一作为源 RCCS(无论选哪一棵映射，效果一样，算法具有对称性)，另一作为目标 RCCS。根据同功能节点的定义查找可映射项，依一定规则将其记录入映射表。

4.3 算法分析与比较

可以验证，在假设 2 的情况下，算法⑤和算法④是等效的。但算法⑤中的成功匹配是在同功能节点映射的条件下进行的，它能排除相关节点但非同功能节点的情况。从这个角度说，它比算法④效果好。另外，此机制能找到弱匹配项。

下面我们来简单比较一下算法④和算法⑤的时间复杂度。

假设：平均每个 RCCS 有 R 个节点，每个节点的 ID 号平均长度为 $\frac{R}{d}$，则字串匹配的时间为 $O(\frac{R}{d}+\frac{R}{d})$(KMP 算法)，最坏情况的时间花费为 $O(2R)$；req 和 msu 从 RCCS 中选择 $\frac{R}{c}$ 个($c>1$)节点，字典的长度为 h(指有 h 组同义词)，对任两个词查字典，看其是否属于同义的时间复杂度为 $O(h\cdot h)=O(h^2)$；

对于算法④，对每一对 req 节点和 msu 节点，因 ID 号平均长 $\frac{R}{d}$，所以其祖孙个数平均为 $\frac{R}{d}$ 个；对一对节点查找它们及其祖孙是否相关节点的时间花费为 $\frac{R}{d}\cdot\frac{R}{d}\cdot h^2$。因此，对每个 req 节点(遍历 msu 节点)的匹配时间是：$\frac{R}{c}\cdot\left(\frac{R}{d}\cdot\frac{R}{d}\right)\cdot h^2$。设有 n 组 req 和 msu 进行匹配，算法④的时间复杂度为：

$$T_4(n,R,h)=n\cdot\left[\frac{R}{c}\cdot\frac{R}{c}\cdot\left(\frac{R}{d}\cdot\frac{R}{d}\right)\cdot h^2\right]=O(R^4h^2n)$$

对于算法⑤，需要按方法分步讨论。

维普资讯 http://www.cqvip.com

第一步，RCCS间的语义映射。设平均有$\frac{R}{a}$个节点在另一棵RCCS上有相关节点，那么对于每个相关节点还要检测其子树，设节点子树节点个数平均为$\frac{R}{b}$个，对于两个节点的子树上的节点都要判断其是否相关，所以对于相关节点的检测时间(由算法的递归关系知最坏遍历子树上所有节点)得：$\frac{R}{a}\cdot\left(\frac{R}{b}\cdot\frac{R}{b}\right)\cdot h^2$；而对于非相关节点只需要遍历目标RCCS上所有节点，查字典判断其是否同义即可，因此对于所有源RCCS节点的检测时间为$\frac{R}{a}\cdot\left(\frac{R}{b}\cdot\frac{R}{b}\right)\cdot h^2+R(1-\frac{1}{a})\cdot R\cdot h^2$。

第二步，将req转换为新req。对每个req查找映射表，映射表最大长度为$\frac{R}{a}$(每个相关节点均可语义映射)，所以时间为：$\frac{R}{c}\cdot\frac{R}{a}$。

第三步，对新req和msu进行匹配，即分析算法①的时间复杂度。对每个req节点(遍历msu节点)所用时间为：$\frac{R}{c}\cdot 2R$；得$T_1(R)=\frac{R}{c}\cdot\frac{R}{c}\cdot 2R=O(R^3)$。

设有n组req和msu进行匹配：$T_5(R,h,n)=\frac{R}{a}\cdot\left(\frac{R}{b}\cdot\frac{R}{b}\right)\cdot h^2+R(1-\frac{1}{a})\cdot R\cdot h^2+n\cdot\left(\frac{R}{c}\cdot\frac{R}{a}+\frac{R}{c}\cdot\frac{R}{c}\cdot 2R\right)=O(R^3\cdot h^2+n\cdot R^3)$

一般来说，字典数据是非常大的，h比R的数量级要高。如果对这种互操作的需求很高，对一个req可以找到若干msu需要与之进行匹配，那么n的增长速度也会非常快。将R看作常数，得：

$T_4(n,h)=O(n\cdot h)$；$T_5(n,h)=O(\max(h,n))$

显然，算法⑤的时间效率要高得多。

5 讨论

本文引入ISO16100国际标准对软件描述的建模方法，研究了基于这种建模方法进行需求匹配的机制。按照标准建模方法和需求分析，分别讨论了简单匹配(req和msu建模于同一RCCS)和语义匹配(req和msu建模于不同RCCS)两种情况。

简单匹配参考同一模板，可以基于ID进行匹配。

对于语义匹配的情况，最简单的就是算法②，它无需参考模板，仅需要查字典，是脱离于标准建模方法也可以进行匹配的一个特例。另外，本文还提供了两种需要参考字典中同义词项的匹配机制：

• 一种是基于能力描述间的节点的功能比较，这种匹配机制基于req的节点展开，在对req和msu的节点匹配时，分别参考不同的模板(RCCS)。

• 另一种是将req转换为新(同义)的能力描述，然后将其与msu进行简单匹配，以达到符合要求的语义匹配。

简单匹配和语义匹配都支持单元匹配级别，对每个req节点都能找到精确匹配、弱匹配和无匹配。其中，精确匹配扩展一般的匹配于功能分解，它不仅可以找出对该需求功能的实现，而且可以找出实现了该功能的更高一层(即集成了该功能)的功能模块；弱匹配可以提示将req此节点进行需求的功能分解(依RCCS)：先找出匹配部分功能的软件，然后开发剩余的功能模块，以提高集成效率。

算法⑤的第2)步在无相关映射表时新建映射表，但若每次注册新RCCS时均建立与其相关的映射表，且在没有映射存在(即映射表为空)时数据库中相关项的uri置空，这样算法⑤就可以结束于映射表uri为空的情况。但是这会面临新的挑战：在RCCS不断增多的情况下，是否需要将新注册的RCCS与现有所有RCCS进行映射？还是可以挑选可能映射的RCCS进行映射，而对于彼此根本不可能存在同功能节点的RCCS直接置空？

虽然需要考虑对遗留系统的兼容，但标准最终的目标是统一，比如对于同节点的功能定义、名称定义的广泛统一。这就要求我们在找出不同的模板(RCCS)后对模板进行规格化和统一性的调整，而将遗留系统融入到新系统，或将不同的系统定义统一，将所有注册的msu按新系统进行相应调整和重新注册(自动化)，这都是今后需要考虑的问题。

参 考 文 献

1 ISO 16100：Industrial Automation Systems and Integration —— Manufacturing Software Capability Profiling for Interoperability
Part 1：Framework
Part 2：Profiling Methodology
Part 3：,Interface protocol and Template
Part 4：Conformance test methods, criteria and reports

2 于玮萍，王茜. 基于能力模型的服务匹配机制的研究与实现. 2005年数据库年会，内蒙古自治区，2005

(上接第252页)

这种分支依赖图不仅能够用于进化测试，而且能应用于传统的基于分支覆盖的软件测试以及程序调试和程序理解。

参 考 文 献

1 Clark J, Dolado J J, Harman M, et al. Reformulating software engineering as a search problem. IEE Proceedings -Software, 2003, 150(3): 161～175

2 Wegener J, Baresel A, Sthamer H. Evolutionary test environment for automatic structural testing. Information and Software Technology, 2001, 43(14): 841～854

3 Jones B, Sthamer H H, Eyres D. Automatic structural testing using genetic algorithms. The Software Engineering Journal, 1996, 11: 299～306

4 Michael C, McGraw G, Schatz M. Generating software test data by evolution. IEEE Transactions on Software Engineering, 2001, 27(12): 1085～1110

5 Liu X, Liu H, Wang B. Unified fitness function calculation rule for flag conditions to improve evolutionary testing. The 20th IEEE/ACM International Conference on Automated Software Engineering. Long Beach, California, USA, Nov. 2005

6 Liu X, Liu H, Wang B. Unifying the fitness function calculation for flag conditions. The 6th Metaheuristics International Conference(MIC'05). Vienna, Austria, Aug. 2005

7 Liu X, Wang B, Liu H. Evolutionary Testing in the context of Object-Oriented Programs. In: Proceeding of the 8th Intl. conf. for Young Computer Scientists (ICYCS'05). Beijing, China, Sept. 2005

8 Rountev A, Kagan S, Gibas M. Static and Dynamic Analysis of call chains in Java. In: Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis, Jul 2004

9 Ottenstein K, Ottenstein L. The program dependence graph in a software development environment. In: Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, 1984. 177～184

10 Zhao J. Applying program dependence analysis to Java software. In: Proc. Workshop on Software Engineering and Database Systems, Dec. 1998. 162～169