第 40 卷 第 11 期 计 算 机 工 程 2014 年 11 月
Vol. 40 No. 11 Computer Engineering November 2014

·体系结构与软件技术· 文章编号：1000-3428(2014)11-0071-06 文献标识码：A 中图分类号：TP31

结合可达性分析的代码片段推荐

贾 翕，于海波，方 璐

（上海交通大学软件学院，上海 200240）

摘 要：为满足日益复杂的软件需求，开发人员需要通过代码提示工具来辅助完成开发任务，但现有代码提示工具在推荐包含静态方法的代码片段时存在空间爆炸问题。为此，提出一种基于程序环境信息的代码片段推荐方法。结合可达性分析进行推荐能够有效削减静态方法入口点，在避免空间爆炸的同时，还可以准确、有效地描述程序环境信息。基于该方法实现在 Eclipse 中的代码推荐插件，并对 Tomcat 源码进行实验验证。实验结果表明，该方法可实现静态方法的代码片段推荐，与 Eclipse Code Recommenders 插件中的推荐方法相比，能获得更准确的推荐结果。

关键词：静态方法；代码片段；语义网规则语言；可达性分析；代码推荐；排序

中文引用格式：贾 翕，于海波，方 璐. 结合可达性分析的代码片段推荐[J]. 计算机工程，2014，40(11)：71-76.

英文引用格式：Jia Xi，Yu Haibo，Fang Lu. Code Snippet Recommendation Combining with Reachability Analysis[J]. Computer Engineering，2014，40(11)：71-76.

Code Snippet Recommendation Combining with Reachability Analysis

JIA Xi，YU Haibo，FANG Lu

（School of Software，Shanghai Jiaotong University，Shanghai 200240，China）

【Abstract】 As the software requirement becoming more and more complicated，developers have an increasing dependency on the code recommendation tools to assist their development tasks，while current code recommendation tools can not provide efficient recommendation for static methods. In order to recommend code snippet including static methods in a faster and more accurate mean，this paper proposes code snippet recommendation method based on program context combing with the reachability analysis，which solves the space explosion problem during the static method recommendation well，and it can also describe the program context accurately and effectively. Based on the proposed method，a code recommendation Eclipse plugin is implemented，and a corresponding experiment on Tomcat source code is conducted. Experimental results show that，the proposed method not only owns the capability of recommending code snippet with static method，but also has a higher accuracy compared with the recommendation method of Eclipse Code Recommenders.

【Key words】 static method；code snippet；Semantic Web Rule Language（SWRL）；reachability analysis；code recommendation；ranking

DOI：10.3969/j.issn.1000-3428.2014.11.014

1 概述

由于软件系统越来越复杂，因此研究者大多采用成熟的框架来进行开发，以此来提高软件开发的效率和质量。目前框架的种类越来越多，而且框架本身也越来越复杂，例如，在.NET Framework 4.0 中有近 700 个命名空间、30 000 个类型、280 000 个方法，在这种情况下，开发人员想要在短时间内掌握庞大的信息就变得十分具有挑战性。因此，开发环境中集成了很多代码提示工具来帮助开发人员进行开发工作，如参数补全、函数提示等。

目前，代码片段推荐主要采用的方法有基于历史代码的推荐方法和基于程序环境信息的推荐方法。但对于包含静态方法的代码片段，这 2 种方法均不能给出满意的推荐结果，其原因在于，仅根据程序上下文，无法得知所有可能的静态成员，而即使得知所有的静态成员，由于其数量众多，如果将其全部加至入口点进行搜索，会造成空间爆炸。

作者简介：贾 翕（1991－），男，硕士研究生，主研方向：语义网；于海波，讲师；方 璐，硕士研究生。

收稿日期：2013-12-11 修回日期：2014-01-02 E-mail：g.jessejia@gmail.com

针对上述问题,本文提出一种基于程序环境信息的代码片段推荐方法,该方法通过本体描述 Java 程序代码,结合 SWRL 规则语言[1]进行类型的可达性分析,同时,根据代码片段的特征对推荐结果进行排序,使推荐结果更加合理。

2 相关工作

目前,代码片段推荐研究中主要采用的方法有 2 种:基于历史代码的推荐方法,基于程序环境信息的推荐方法。

基于历史代码的推荐方法工作较多,其中,文献[2]提出了基于历史代码的推荐方法,其主要思想为根据当前开发环境的上下文,在历史代码中寻找与之相似的节点,查看历史代码中的代码片段并推荐给开发人员。在后续的工作中,扩大了历史代码的范围,采用了网络上的开源代码[3];通过挖掘用例模型来提高推荐的准确性[4-5];以及采用贝叶斯网络进行概率分析[6]。但是在该类方法中,历史代码的质量直接影响到推荐结果,因此,该方法主要存在以下 2 个方面的问题:(1)无法确保历史代码的覆盖范围,当没有相应的历史代码时将无法做出推荐;(2)面对日益更新的框架,无法确保历史代码是否过时,甚至是否正确。

文献[7-8]提出了基于程序环境信息的合成方法,其主要流程包括以下 4 个部分:

(1) 根据程序上下文选取合适的入口点,如局部变量,当前可直接访问的方法等。

(2) 对于每一个入口点,根据其类型中可访问的类型转换(通过访问域成员或者方法成员得到另外一个类型的实例),逐层展开,构成一棵类型转换树。

(3) 在类型转换树上进行搜索,找出根节点到目标类型的可达路径。

(4) 对这些可达路径进行排序,将结果返回给开发人员。

在上述方法中,通常树的展开和搜索是同时进行的,而且,为了限制搜索空间,通常会限制搜索的深度或者可能路径的最大数目,当搜索范围超过限制的深度或者搜索结果已经找到足够多的路径时,就停止搜索。

图 1 展示了在 Eclipse 环境下进行开发过程的一个场景,开发人员在开发 Eclipse 插件时,希望修改当前状态栏的信息。开发人员通过查阅 API 文档后,找到了 IStatusLineManager 接口下的 setMessage 方法。但当前的开发环境并未告知开发人员如何获取一个 IStatus LineManager 的实例。本文针对上述场景,研究如何有效地进行推荐出形如图 1 所示的 getViewSite( ). getActionBars( ). getStatus LineManager( )代码片段,并使推荐结果和上下文紧密相关。

图 1 代码推荐应用场景实例

对于图 1 中所示的静态方法的代码片段,现有方法并不能给出满意的推荐结果,其原因在于:仅根据程序上下文,无法得知所有可能的静态成员,而即使得知所有的静态成员,由于其数量众多,如果将其全部加至入口点进行搜索,会造成空间爆炸。

文献[9-11]使用了本体对代码进行描述,但是其不能很好地适用于类型转换问题。

为解决含静态方法的代码片段推荐中的空间爆炸问题,对含静态方法的代码片段给出较好的推荐结果,本文提出一种基于程序环境信息的代码片段推荐方法,该方法通过本体描述 Java 程序代码,结合 SWRL 规则语言[1]进行类型的可达性分析,同时,根据代码片段的特征对推荐结果进行排序,使推荐结果更加合理。

3 工作流程

为了解决本文第 2 节中所述静态方法推荐过程中的空间爆炸问题,需要削减静态入口点的数目。本文研究通过计算类型的可达性来选取可能的入口点,使其在推荐过程中,仅对目标类型可达的静态方法和成员进行搜索。

类型可达性是指:在一个特定的程序环境下,一种类型能否通过一系列的类型转换得到目标类型,它是由一系列类型转换的传递决定的。类型可达性分析的难点在于:

(1) 类型转换并不是在所有环境下都可以一概而论,在实际应用中,很多类型转换具有可见性(如 Java 语言中的 protected 和 private 关键字),从而导致在不同的程序环境下,类型转换可能具有不同的可达性。

(2) 在类型转换具有不同的可见性时,可达性不具备明显的传递性。

因此,为了准确地分析可达性,需要提出一种准确计算可达性的方法。而可达性又和当前程序环境相关联,所以本文采用一种形式化的方法,首先构建了一个简单的本体来描述程序的基本信息,并基于 SWRL 规则语言定义了类型转换及可达性判断规则,然后通过 pellet 推理机[12]对基本信息进行推理,计算出与环境无关的可达性信息;而与环境相关的可达性信息,本文通过有限次数的对与环境无关的可达性信息的查询计算出结果。

本文所提出的结合可达性分析的代码片段推荐方法工作流程如图 2 所示。该工作流程分为预计算和实时计算 2 个部分。预计算对当前工程所引入代码进行信息提取,通过 SWRL 规则推理得到可达性。在推荐过程中,可达性信息将被用来判断目标类型是否可达。在实时计算部分,当开发人员在开发环境中发起推荐请求时,根据程序上下文以及可达性信息,本文方法将得出搜索的入口点,通过搜索得出的多个代码片段再结合代码片段特征及程序上下文排序后,将作为最终的推荐结果返回给开发人员。

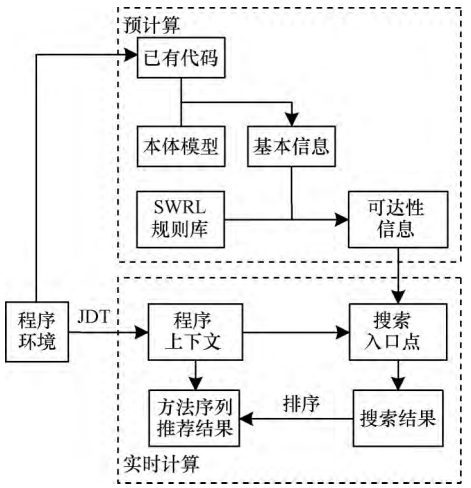

图 2　含静态方法代码片段的推荐过程

4　可达性分析及推荐方法

4.1　Java 类型转换本体模型定义与基本信息生成

在 Java 类型的本体模型中主要有 4 个部分组成,分别是类、对象属性、数据属性和规则。类定义了 Java 类型、方法等具体概念以及它们本身的特性;对象属性定义了这些概念之间存在的关系,也就是类型和方法具备什么样的关联;数据属性则描述了本体类具备的一些常量信息,例如类和方法的名称等;规则定义了如何根据一系列关系和概念推理出新的关系和概念,通过规则可以推导出类型转换及类型可达关系。

图 3 描述了 Java 类型转换的本体模型的主要部分,其中本体和对象属性描述如下:

本体类 package 用来描述 Java 语言中的包;本体类 Type 用来描述 Java 语言中的类型,本体类 Type 下包含的2 个子类:Clazz 和 Interface,分别用来描述 Java 语言中类和接口;本体类 Member 用来描述 Java 语言中的成员,本体类 Member 下包含的2 个子类:Method 和 Field,分别用来描述 Java 语言中的成员方法和成员变量;本体类 Visibility 用来描述 Java 语言中成员的可见性,其 4 个实例对应 Java 语言中的 4 个关键字,分别为 Public,Protected,Default 和 Private。

对象属性 HasReturnType 用来描述 Java 语言中成员的类型,其中成员变量的类型为声明类型,而成员方法的类型为该方法的返回类型;对象属性 IsA 表示了 Java 语言中类型的继承关系,其不仅表示了类型的继承关系,还表示了接口的实现,对象属性 IsA 关系具有传递性;对象属性 Transfer 来表达类型转换关系,根据 2 种不同的可见性,有 2 种不同的类型转换关系,其含义如下:

PublicTransfer($T1$, $T2$):在任何环境中,$T1$ 都可以通过类型转换到达类型 $T2$;

ProtectTransfer($T1$, $T2$):必须在 $T1$ 的子孙类中,$T1$ 可以通过类型转换到达类型 $T2$;

DefaultTransfer($T1$, $T2$):必须在 $T1$ 的同包类中,$T1$ 可以通过类型转换到达类型 $T2$;

PrivateTransfer($T1$, $T2$):必须在 $T1$ 类中,$T1$ 可以通过类型转换到达类型 $T2$。

对象属性 Reach 表达的是类型可达关系,根据 4 种不同的可见性,与对象属性 Transfer 一致,也有 4 种可达关系:PublicReach,ProtectReach,DefaultReach 和 Private-Reach。

依据上述本体描述和程序源码,可以得到类型、成员、可见性等基本信息。

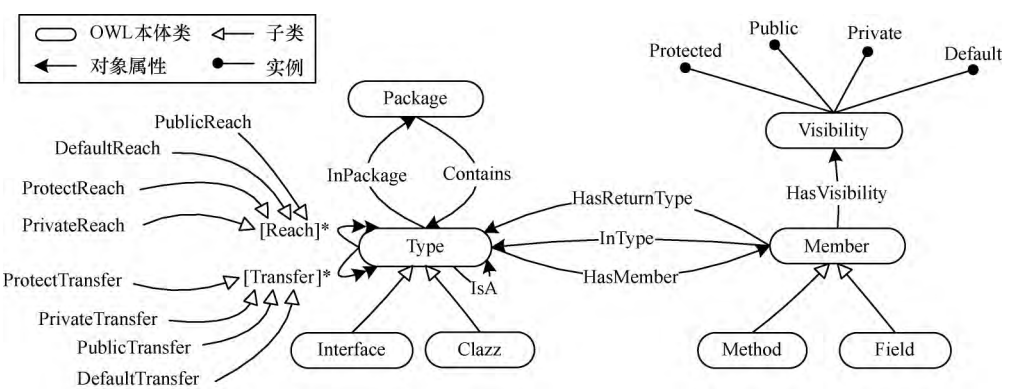

图 3　Java 类型转换的本体模型

4.2 SWRL 规则定义与可达性信息生成

SWRL 规则分为 2 类:第 1 类规则定义了如何基于类型自身属性直接推导得到相应的类型转换,即图 4 中的 4 条规则;第 2 类规则定义了如何基于类型转换信息推导出相应的类型可达性,将在下文中详细阐述。通过图 3 类型转换的 SWRL 规则和 4.1 节中得到的程序基本信息,就可以推理得到代码中所包含的类型转换信息。通过图 5 类型可达的 SWRL 规则中的规则 5 ~ 规则 8,可以得到最简单的可达性信息。规则 9 ~ 规则 11 表述的是类型可达性与环境无关的传递性,所以,这些信息可以通过预计算来得出。

通过上述规则,即可得到与环境无关的类型可达性信息。

规则1	由成员的可见性和返回类型得到4种不同的类型转换	HasMember (?A,?M) ∧HasReturnType(?M,?B) ∧HasVisibility(?M,P ublic)→PublicTransfer(?A,?B)
规则2		HasMember (?A,?M) ∧HasReturnType(?M,?B) ∧HasVisibility(?M, Protected)→P rotect Transfer(?A,?B)
规则3		HasMember (?A,?M) ∧HasReturnType(?M,?B) ∧HasVisibility(?M,Private)→PrivateTransfer(?A,?B)
规则4		HasMember (?A,?M) ∧HasReturnType(?M ,?B) ∧HasVisibility(?M, Default)→DefaultTransfer(?A,?B)

图 4 类型转换的 SWRL 规则

规则5	由4种不同的类型转换得到类型可达性	PublicTransfer(?A,?B)→PublicReach(?A,?B)
规则6		ProtectTransfer(?A,?B)→ProtectReach(?A,?B) ; ProtectTransfer(?A,?B)→DefaultReach(?A,?B)
规则7		PrivateTransfer(?A,?B)→PrivateReach(?A,?B)
规则8		DefaultTransfer(?A,?B)→DefaultReach(?A,?B)
规则9	类型可达性的publicReach后继传递性	[Protect,Private,Public,Default,]Reach(?A,?B)∧PublicReach(?B,?C)→[Protect,Private,Public,Default,]Reach(?A,?C)
规则10	子类可达关系的传递性	ProtectReach(?A,?B)∧ProtectReach(?B,?C)∧IsA(?A,?B)→ProtectReach(?A,?C)
规则11	同包可达关系的传递性	DefaultReach(?A,?B)∧DefaultReach(?B,?C)SamePackage(?A,?B)→DefaultReach(?A,?C)

图 5 类型可达的 SWRL 规则

4.3 入口点选取

根据 4.2 节中的可达性信息生成方法,可以直接得到与环境信息无关的静态成员入口点,这些静态成员的返回类型可达目标类型,剩余的一部分可用的静态成员入口点,则需要结合当前程序环境的信息进行查询。

发起请求时,当前环境类型为 E,其祖先类型的集合为 $A(T)$,与其在同一包下的类型集合为 $S(T)$,目标类型为 D,那么一个类型 T 在环境类型 E 中可达类型 D 必须满足以下条件中的一个:

(1) T 在当前环境类型 E 的限制下可达 $A(T)$ 中的任一类型 A,且 A 和 D 具有 ProtectReach 关系。

(2) T 在当前环境类型 E 的限制下可达 $S(T)$ 中的任一类型 S,且 S 和 D 具有 DefaultReach 关系。

(3) T 和 E 为同一类型,且 E 和 D 具有 PrivateReach 关系。

通过以上方法就可以选取出可用的静态成员入口点,并结合当前程序环境中可以访问的域成员、方法以及局部变量,由这 4 个部分构成搜索的入口点。

4.4 搜索

搜索过程的具体步骤如下:

(1) 预定义一个 COST 常量,其为当前代码片段最大代价值。

(2) 在口点中进行搜索不超过目标 COST 的代码片段,若返回值为需要类型的代码片段,这些将计入结果;若其代价超过了目标 COST,这些代码片段将作为下一次搜索的入口点;若既不是笔者需要的类型,也没有超过目标 COST,这些代码片段将被抛弃。

(3) 如果没有找到足够多的代码片段,则提高目标 COST,用第(2)步产生的入口点,进行下一次搜索;若第(2)步没有产生更多的入口点或已找到足够多的代码片段,搜索结束。

4.5 排序

代码片段的代价为每一步类型转换的代价总和,对于每一步类型转换的代价,本文通过以下因素计算得出:

(1) 代码片段长度。代码片段的长度即一个代码片段完成所需要的类型转换的数目,笔者希望以更少的类型转换来达到目标类型,所以,在代价计算中,若代码片段长度为 *length*,最后代码片段的整体代价增加 C_{length}。

(2) 类型转换类型。在实际应用中,笔者更希望使用对象方法调用,而不是其他类型下的域成员或者是静态方法作为其中的一环,因为静态方法是全局可见的,对象方法比静态方法与上下文关系更加紧密,而直接访问其他类型的域成员是一个坏的编程方式。若代码片段中访问其他类型的域成员次

数为 Num_{field},使用静态方法次数为 Num_{static},代码片段的整体代价增加 $Num_{field} \times C_{field} + Num_{static} \times C_{static}$。

(3) 方法调用中的参数类型。程序上下文能提供一部分信息,这些信息可以被用作填充代码片段中所需要的参数。如果一个代码片段的参数类型在当前环境能够提供,说明该代码片段与当前环境关系紧密。在计算中,若代码片段使用能由当前环境提供的参数次数为 $Num_{provided}$,而使用了当前环境不能提供的类型的次数为 $Num_{unprovided}$,代码片段的整体代价增加 $Num_{provided} \times C_{provided} + Num_{unprovided} \times C_{unprovided}$。

通过在已有的历史代码中进行机器学习,每个因素权重取值如表 1 所示时,可取得较高的命中率。

表 1　因素权重取值

因素名称	取值
C_{length}	1
C_{field}	2
C_{static}	4
$C_{provided}$	1
$C_{unprovided}$	4

5　代码片段推荐实验

5.1　实验环境与方法

为验证可达性在寻找静态方法入口的有效性,以及推荐排序方法的准确性,本文进行实验数据分析。本次实验的开发环境为 Eclipse 4.2,实验设备为一台内存为 8 GB,CPU 为酷睿双核 2.5 GHz,操作系统为 Windows 7 的 PC 机。通过扩展 Eclipse Code Recommenders 提供的扩展点,实现了在 Eclipse 中的代码推荐插件,并在 Eclipse4.2 中,下载并编译了 Tomcat 7.0.47 的源码,在源码中找出了 1 338 处获取类型实例的代码片段,其中 145 处包含了静态方法入口点,对这些代码片段的调用点分别使用基于本文方法的插件和 Eclipse Code Recommenders 中的推荐功能[12],与源码进行对比统计,以该统计结果分析方法的准确性和有效性。

5.2　实验结果与分析

在 Tomcat 7.0.47 的源码中,共找出 7 处静态成员作为起始的获取类型实例的代码片段,在 Tomcat 及所依赖的类库(包括 JRE)中共有 7 451 处静态成员入口点,在可达性分析后入口点数的统计如表 2 所示,可以看出在可达性分析后,静态入口明显减少,其中在 6 个入口点以下的达到 81.3%,可以有效地削减搜索空间。

表 2　静态方法入口点削减结果

削减后入口点数目	命中频次
1	3
2	11
3	25
4	32
5	34
6	13
>6	27

在这 145 处静态方法推荐中,命中率如表 3 所示,可以看出,在削减了搜索空间后,依然能保证 78% 的推荐命中率,说明可达性分析是准确的。

表 3　静态方法代码片段命中频次

代码片段命中次序	命中频次
1	12
2 ~ 3	66
4 ~ 10	35
未命中	32

在 Tomcat 7.0.47 的源码中,共有 1 338 处获取类型实例的代码片段,实验结果如表 4 所示,可以看出,大部分情况下都可以在推荐列表中的前三位命中,命中率达到 72.2%,其中前十位达到了 90.3%,准确率较高,相较目前 Eclipse 中的 Eclipse Code Recommenders,在前三命中率(60.7%)和整体命中率(78.4%)上均有所提高,原因在于 145 处包含静态方法的代码片段中,本文方法能有效推荐其中 112 处,提高整体命中率 8.4%;此外,更合理的排序也使得命中率有所提高。

表 4　代码片段命中频次比较

代码片段命中次序	命中频次	
	本文方法	Eclipse Code Recommenders
1	231	194
2 ~ 3	735	618
4 ~ 10	242	237
未命中	130	289

进一步分析结果,在未命中 130 处中,有 92 处属于向下类型转换,无法给出合理的推荐,对于向下类型转换,由于目前的推荐算法是基于静态代码,无法很好地预知程序执行过程中的实际类型,因此较难做出合理的推荐。

6 结束语

本文提出一种结合可达性分析实现代码片段推荐的方法,在排序时结合了程序内部信息,增强了语义性。实验结果表明,可达性分析在处理静态方法推荐时可有效削减搜索空间,在扩大可推荐代码片段范围的同时,也可提高推荐结果的准确率。与传统推荐方法相比,本文方法推荐的可用范围更广,该方法不仅可用于对象方法的序列,而且对静态方法也能进行准确推荐;同时在获取多个结果时,能对结果进行有效排序,提供给开发人员更直观、有效的结果。但由于本文方法是基于静态的程序上下文信息,无法很好地应对程序的动态特性,因此对其进行改进是下一步的研究方向。

参考文献

[1] Horrocks I, Patel-Schneider P F, Boley H, et al. SWRL: A Semantic Web Rule Language Combining OWL and RuleML[J]. W3C Member Submission, 2004, 21: 79.

[2] Holmes R, Murphy G C. Using Structural Context to Recommend Source Code Examples[C]//Proceedings of the 27th International Conference on Software Engineering. [S.l.]: ACM Press, 2005: 117-125.

[3] Thummalapenta S, Xie Tao. Parseweb: A Programmer Assistant for Reusing Open Source Code on the Web [C]//Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering. [S.l.]: ACM Press, 2007: 204-213.

[4] Xie Tao, Pei Jian. MAPO: Mining API Usages from Open Source Repositories[C]//Proceedings of 2006 International Workshop on Mining Software Repositories. [S.l.]: ACM Press, 2006: 54-57.

[5] Mcmillan C, Poshyvanyk D, Grechanik M, et al. Portfolio: Searching for Relevant Functions and Their Usages in Millions of Lines of Code[J]. ACM Tran-sactions on Software Engineering and Methodology, 2013, 22(4): 37.

[6] 章 程. 基于机器学习和程序分析相结合的程序调试技术研究[D]. 上海: 上海交通大学, 2013.

[7] Perelman D, Gulwani S, Ball T, et al. Type-directed Completion of Partial Expressions[J]. ACM SIGPLAN Notices, 2012, 47(6): 275-286.

[8] Gvero T, Kuncak V, Kuraj I, et al. Complete Completion Using Types and Weights [C]//Proceedings of PLDI'13. [S.l.]: ACM Press, 2013: 27-38.

[9] Würsch M, Ghezzi G, Hert M, et al. SEON: A Pyramid of Ontologies for Software Evolution and Its Applications[J]. Computing, 2012, 94(11): 857-885.

[10] Keivanloo I, Rilling J, Charland P. Internet-scale Real-time Code Clone Search via Multi-level Indexing [C]// Proceedings of WCRE'11. [S.l.]: IEEE Press, 2011: 23-27.

[11] Frey T. Hypermodelling: Next Level Software Engineering with Data Warehouses[EB/OL]. (2013-06-26). http://wwwiti.cs.uni-magdeburg.de/iti_db/publikationen/ps/auto/phdFrey13.pdf.

[12] Sirin E, Parsia B, Grau B C, et al. Pellet: A Practical OWL-DL Reasoner[J]. Journal of Web Semantics, 5(2): 51-53.

编辑 金胡考

(上接第 70 页)

5 结束语

近年来,面向 Web 服务的访问控制研究备受关注,然而大多数都是针对集团型企业应用系统,不能很好地适应复杂电子政务系统。本文在研究基于组织的四层访问控制模型(OB4LAC)的基础上,提出了基于组织的 Web 服务访问控制模型(WS-OBAC)。

该模型的优势主要表现在:(1)基于组织岗位的授权模式与复杂电子政务系统的组织结构契合,用户与角色相分离,岗位与授权相关联,更加符合复杂电子政务建设的实际需求。(2)通过引进岗位代理和授权单元,根据访问服务的环境上下文和相关约束条件,动态控制用户的实际可执行权限。并通过授权单元的状态迁移,对面向组合 Web 服务业务流程的访问控制提供支持,促进政府机构通过 Web 服务技术对业务流程进行重组,实现各机构、各部门之间互通互联和政务协作。(3)将访问权限分为服务权限和服务属性权限,用户必须同时拥有某一服务的服务权限和对应的服务属性权限,才能够与服务进行交互,从而更好地保护服务资源。(4)易于实现统一授权管理、单点登录和多部门系统集成。

参考文献

[1] 文俊浩,饶锡如,何 盼,等. 基于工作流的服务组合在电子政务中的应用[J]. 计算机应用, 2009, 29(9): 2512-2515.

[2] 邓集波,洪 帆. 基于任务的访问控制模型[J]. 软件学报, 2003, 14(1): 76-82.

[3] 许 峰,赖海光,黄 浩,等. 面向服务的角色访问控制技术研究[J]. 计算机学报, 2005, 28(4): 686-693.

[4] 朱一群,李建华,张全海. 一种面向 Web 服务的动态分级角色访问控制模型[J]. 上海交通大学学报, 2007, 41(5): 783-788.

[5] 上超望,刘清堂,赵呈领,等. UCON 支持的组合 Web 服务业务流程访问控制模型[J]. 武汉大学学报:理学版, 2011, 57(5): 408-412.

[6] Park J, Sandhu R. The $UCON_{ABC}$ Usage Control Model [J]. ACM Transactions on Information and System Security, 2004, 7(1): 128-174.

[7] 刘 淼,李 鹏. 结合属性和角色的 Web 服务访问控制[J]. 计算机工程与设计, 2012, 33(2): 484-488.

[8] 张 斌,张 宇. 基于属性和角色的访问控制模型[J]. 计算机工程与设计, 2012, 33(10): 3807-3811.

[9] 张 帅,孙建伶,徐 斌,等. 基于 RBAC 的跨多企业服务组合访问控制模型[J]. 浙江大学学报:工学版, 2012, 46(11): 2035-2043.

[10] Sandhu R. Role-based Access Control Models[J]. IEEE Computer, 1996, 29(2): 38-47.

[11] 廖俊国,洪 帆,肖海军,等. 细粒度的基于角色的访问控制模型[J]. 计算机工程与应用, 2007, 43(34): 138-140.

[12] 李怀明. 电子政务系统中基于组织的访问控制模型研究[D]. 大连: 大连理工大学, 2009.

编辑 任吉慧