基于本体的构件化软件演化信息获取及度量研究

钟林辉 宗洪雁
(江西师范大学计算机信息工程学院 南昌 330022)

摘 要 软件演化信息是一种重要的、能反映软件变化历史的信息。然而,传统的软件演化信息以文件或者项目作为跟踪软件变化的基本单元,不能有效地支持构件化软件演化信息的存储和检索。提出了采用本体概念表示构件化软件演化信息的策略,并利用Jena推理机实现构件化软件演化信息的获取。该方法不仅能检索构件化软件的基本演化信息,而且可以通过定义规则的方式检索出蕴含的演化信息。同时,文中也提出了一种构件化软件演化度量的模型,该模型通过对演化属性的计算分析来预测构件化软件的演化趋势。

关键词 构件化软件,软件演化,本体,演化度量
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2015.1.044

Research on Evolution Information Acquisition and Measurement of Component-based Software Based on Ontology Model

ZHONG Lin-hui ZONG Hong-yan
(School of Computer Information and Engineering, Jiangxi Normal University, Nanchang 330022, China)

Abstract Software evolution is important information reflecting the software change history. However, traditional software evolution information caption methods use the file or project as the basic unit to track the software change, which cannot effectively support the storage and retrieval of component-based software evolution information. This paper presented the strategies of modeling the component-based software evolution information based on the ontology model, and used the Jena inference engine to acquire the software evolution information. This method can not only query the basic software evolution information directly, but also retrieve the software evolution information by defining the rules. In addition, this paper proposed a component-based software measurement model, which can be used to forecast evolution trend by analyzing the evolution properties of the component-based software.

Keywords Component-based software, Software evolution, Ontology, Software evolution measurement

1 前言

软件演化是软件不断更新变化的过程,是软件的本质特征之一。当前对软件演化的研究越来越受到重视。早在20世纪70年代,Lehman等人就通过研究软件的变化历史信息,发现了软件演化的规律[1];近几年,软件工程国际大会连续召开了几次关于"挖掘软件资产库(mining software repository)"的会议,旨在讨论如何通过挖掘软件资产库,理解软件演化的规律并加以合理运用,以此达到提高软件质量、开发高可靠性软件的目的。

然而,目前软件演化信息主要来源于软件配置管理系统、错误报告系统等CASE工具。这些系统在设计时主要是以文件或者项目为基本单元,来记录软件的变化历史。因此,在软件演化信息的检索上存在以下几个方面的问题:①缺乏对较高层次软件系统的演化支持。例如对于构件化软件,构件和软件体系结构等概念不能直接地映射和体现在软件配置管理系统中,这导致在处理构件化软件演化信息时更加复杂。②缺乏对软件演化信息的推理能力。一般而言,软件配置管理系统记录了软件的版本、修改原因等演化信息,通过简单的检索就能够实现对这些软件演化信息的提取。但是,对于较为复杂的软件演化信息例如变化度量等则无能为力。

为了解决上述问题,本文以构件化软件演化信息为研究对象,在文献[23,24]前期研究的基础上,利用本体概念及本体描述语言,描述构件化软件演化这一特定领域中的概念及概念间的关系;并利用Jena推理机实现对构件化软件演化信息的检索。本文第2节介绍软件演化信息建模与获取方法方面的相关研究;第3节介绍基于本体的构件化软件演化信息模型及表示;第4节讨论了基于Jena推理机的构件化软件演化信息获取,并提出了一种构件化软件演化度量方法;最后对相关工作进行总结。

2 相关研究

软件演化信息是软件变化过程中存储或蕴含的有用信息。一般,软件演化信息的主要来源是软件配置管理系统或

到稿日期:2014-01-13 返修日期:2014-08-19 本文受国家自然科学基金项目(61262015,61462040),江西省自然科学基金项目(20142BAB207027,20142BAB207011),江西省教育厅科学技术项目(GJJ13230)资助。

钟林辉 男,博士,副教授,主要研究领域为构件化软件、软件体系结构、软件自动化、软件演化,E-mail:shiningto@sohu.com;宗洪雁 男,硕士生,主要研究领域为软件演化、软件形式化。

者软件版本系统，例如 CVS[4]，Subversion[5] 等。可以将软件演化信息的获取方式分为两类：

(1) 直接演化信息的获取：目前绝大多数的软件配置管理系统都是与具体编程语言无关的，以文件或者目录作为记录变化的基本单位，因此能查询出与文件/目录变化有关的版本、分支、时间等演化信息[6]。但是，它们无法跟踪和记录源程序在编程元素和程序结构上的变化。为了克服这个问题，Tudor 等人提出了 Hismo 模型，Hismo 模型是一种以软件变化历史为中心的软件演化元模型，该模型能刻画不同软件实体及其组成部分的变化，并为软件演化的可视化和度量提供基础[2,3]。有的研究者则提出了在软件开发环境中跟踪源程序元素变化的方法。例如在文献[7]中利用聚合的概念能将不同的软件产品联系起来，以便实现统一的变化管理。而 Danny Dig 等设计了一个重构敏感(refactoring-aware)的软件配置管理系统 MolhadoRef，其不仅能有效地解决重构过程中的合并冲突问题，而且通过记录程序的变化历史能更好地理解程序演化[8]。

(2) 蕴含演化信息的获取：蕴含演化信息指需要采用逆向工程方法才能获取到的信息，例如挖掘软件版本间的重构操作。Romain Robbes 提出了挖掘基于变化的软件资产库的思想，将程序表示成抽象语法树 AST，通过记录 AST 上的变化操作(例如增加、删除和属性修改等)来推导程序的重构操作[9]。特别地，Z. Xing 等人在软件设计层次提出了结构差异性算法 umlDiff，通过分析两个不同版本的 UML 文档，可以发现 UML 上实施的重构操作[10]。Dig 等则采用语法和语义相结合的策略检测出 Java 程序不同版本间可能涉及到的重构操作[11]。在 Dig 工作的基础上，Kunal 提出了多种启发式信息用于检测 API 层次的重构操作[12]。Prete 实现了一个基于逻辑的查询工具 REF-FINDER，用于发现两个版本间涉及到的重构操作[19]。Reinout 等则实现了一个历史查询工具，其能够发现多个版本间的重构操作[18]。

在演化信息的获取研究中，有研究者试图将本体的概念运用到软件演化的研究中来，其中国外典型代表是 Christoph Kiefer 在分析软件系统的演化时选择 Web 本体语言(OWL)作为软件资源库的数据交换格式，提出了一种扩展的 RDF 数据模型查询语言 SPARQL 的查询引擎 iSPARQL，用于软件演化的可视化、软件度量以及不良代码的发现等[13]。国内学者对本体在软件工程中的应用进行了大量的研究，其中与软件演化有关的研究主要包括在文献[20]中采用 OWL 对软件工程数据进行建模，以实现对各类软件工程数据进行统一管理；在文献[21]中提出了一种基于本体的可信软件演化框架；在文献[22]中设计了一种基于本体的特征模型演化一致性的检测方法。但是这些方法主要是针对基于本体的数据建模以及一致性检测，对蕴含演化信息的推理以及度量考虑得较少。

3 构件化软件演化信息的本体模型

3.1 构件化软件演化信息的核心数据

为了支持构件化软件演化，文献[23]中提出了基于构件的软件配置管理模型。该模型能较好地适应基于构件的软件开发、追踪和捕获构件(原子构件和复合构件)的演化历史。该模型涉及的主要元素如下。

构件：构件是可以被多个软件系统所复用的具有独立功能的系统构成成分，这是构件的逻辑概念。在实际的物理存储中，构件表现为一组与一个逻辑概念密切相关的文件。构件的组成文件需要按一定的目录结构进行组织，因此配置管理系统中的构件可以定义为通过目录结构组织起来的一组密切相关文件的集合。这个构件概念支持各种形态的构件。

构件版本和构件版本树：构件在开发或修改过程中可能出现多个版本，这些版本记录了构件开发或修改的历史。构件可能同时存在多个开发流，构件的一个版本分支代表一个开发流，每个分支包含多个版本。因此，构件的所有版本不再表现为版本的序列，而是构成一个树型结构，称为构件版本树。构件的版本控制通过构件的检出和检入操作完成。构件版本树记录了构件的演化情况。构件的演化包括两个方面：构成构件的文件集合变化、结构变化。

配置：配置是指一组配置项的集合，其中每个配置项可以是一个构件，也可以是一个配置(作为配置项的配置也称子配置)，配置具有自包含性。配置可以表示基于构件的软件开发中的复合构件，也可以表示组装出来的系统。

配置的基线(baseline)：为配置及其所有子配置中的构件都选定一个特定版本，就得到了配置的一个基线，该操作称为基线操作。配置的基线表示复合构件或系统的一个版本。

3.2 基于 OWL 的构件化软件演化信息表示

第 3.1 节中概念所对应的部分本体结构如图 1 所示，记为 $C_i\_V_j$ 的形式，即构件 C_i 的第 j 个版本；基线记为 $SA_i\_B_j$，即软件系统 SA_i 的第 j 个基线。

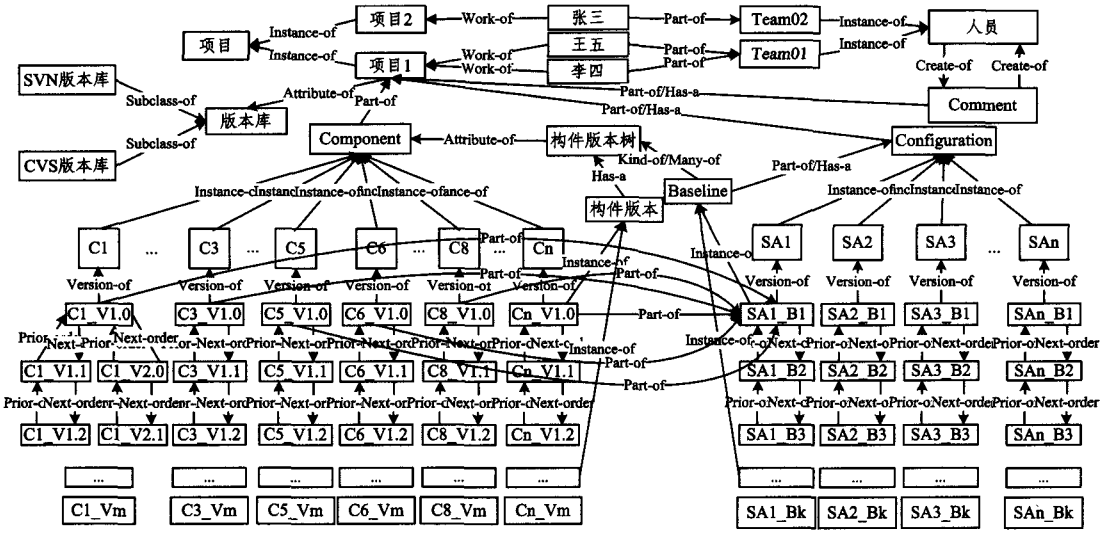

图 1 构件化软件演化信息的本体结构图(局部)

其中，Prior-order 和 Next-order 表示版本之间前驱和后继的关系，例如对于构件 C_1，其版本集合为 $C_1\_V_{1.0}$，$C_1\_V_{1.1}$，$C_1_V_{1.2}$等。$C_1_V_{1.0}$是 $C_1_V_{1.1}$的上一个版本，则存在关系 Prior-order($C_1\_V_{1.1}$，$C_1\_V_{1.0}$)；反之，则存在关系 Next-order ($C_1\_V_{1.0}$，$C_1\_V_{1.1}$)。同理，对于图 1 中的软件系统 SA_i，亦存在基线 $SA_i\_B_j$ 和基线 $SA_i_B_{j+1}$之间的 Prior-order 和 Next-order 关系。同时，为了便于对演化信息的推理，本文使用 OWL-DL[16]作为构件化软件演化信息的描述语言，例如版本树、分支和基线的相关本体描述如下：

• 构件 C_1 的版本树和分支 Branch 的 OWL 本体描述如下：

```
〈!-- http://www.co-ode.org/ontologies/ont.owl#C1 --〉
    〈NamedIndividual rdf:about="&ont;C1"〉
    〈rdf:type rdf:resource="&ont;Component"/〉
    〈ont:Name〉〈/ont:Name〉
    〈ont:Component_id〉〈/ont:Component_id〉
〈rdfs:isDefinedBy rdf:resource="&ont2;Team"/〉
  〈ont2:Part-of rdf:resource="&ont;C1_Branch01"/〉
  〈ont2:Part-of rdf:resource="&ont;C1_Branch02"/〉
  〈ont2:Part-of rdf:resource="&ont;C1_Branch03"/〉
  〈ont2:Part-of rdf:resource="&ont;C1_History"/〉
  〈ont2:Part-of rdf:resource="&ont;C1_V1.0"/〉
  〈ont2:Part-of rdf:resource="&ont;C1_V1.1"/〉
  〈ont2:Part-of rdf:resource="&ont;C1_V1.2"/〉
  〈ont2:Part-of rdf:resource="&ont;C1_V2.0"/〉
  〈ont2:Part-of rdf:resource="&ont;C1_V2.1"/〉
  〈ont2:Part-of rdf:resource="&ont;C1_V2.2"/〉
  〈ont2:Part-of rdf:resource="&ont;C1_V2.3"/〉
  〈ont2:Part-of rdf:resource="&ont;C1_V3.0"/〉
  〈ont2:Part-of rdf:resource="&ont;C1_V3.1"/〉
〈/NamedIndividual〉
〈!-- http://www.co-ode.org/ontologies/ont.owl#C1_Branch01
--〉
    〈NamedIndividual rdf:about="&ont;C1_Branch01"〉
    〈rdf:type rdf:resource="&ont;Branch"/〉
      〈ont:Branch_id〉〈/ont:Branch_id〉
      〈ont2:Part-of rdf:resource="&ont;C1_V1.0"/〉
      〈sameAs rdf:resource="&ont;C1_V1.0"/〉
      〈ont2:Part-of rdf:resource="&ont;C1_V1.1"/〉
      〈sameAs rdf:resource="&ont;C1_V1.1"/〉
      〈ont2:Part-of rdf:resource="&ont;C1_V1.2"/〉
      〈sameAs rdf:resource="&ont;C1_V1.2"/〉
〈/NamedIndividual〉
```

分支 Branch02、Branch03 的约束定义与 Branch01 类似。

• 基线 Baseline 的 OWL 本体表示

在图 1 中，SA_1 的基线 $SA_1\_B_1$ 由构件 C_1 的构件版本 $C_1\_V_{1.0}$、构件 C_3 的构件版本 $C_3_V_{1.0}$以及构件 C_5 的构件版本 $C_5_V_{1.0}$等组成，则基线 $SA_1\_B_1$ 的本体描述：

```
〈!-- http://www.co-ode.org/ontologies/ont.owl# SA1_B1--〉
〈NamedIndividual rdf:about="&ont;SA1_B1"〉
    〈rdf:type rdf:resource="&ont;Baseline"/〉
    〈rdf:type rdf:resource="&ont;Distribute_Baseline"/〉
    〈ont:Baseline_id〉〈/ont:Baseline_id〉
    〈ont2:Part-of rdf:resource="&ont;C1_V1.0"/〉
    〈ont2:Part-of rdf:resource="&ont;C3_V1.0"/〉
    〈ont2:Part-of rdf:resource="&ont;C5_V1.0"/〉
    ……
〈/NamedIndividual〉
```

4 基于本体的构件化软件演化信息获取及度量

4.1 基于本体构件化软件演化信息获取

本节分别介绍构件化软件直接演化信息及蕴含演化信息的获取方法。

4.1.1 直接演化信息获取

针对构件化软件演化信息的本体模型，可以采用多种查询方式实现对构件、基线、版本等信息的检索。例如以图 1 所描述的构件化软件演化信息本体模型为例，在 Protégé[14]中可以借助基于关键字匹配的查询、描述逻辑的 DL Query 查询以及基于属性相关性的 SPARQL 查询机制实现对演化信息的直接获取。

(1)基于关键字匹配的查询方式，获取的信息主要包括概念型和实例型两类。其中，概念型包含与构件或者系统相关的概念如 Component，Configuration，Baseline，Version，Component_history，SA_history，Team 以及各个概念的 URI 标识信息等；实例型则包括版本、分支以及各个实例的 URI 标识信息等。一般，采用基于关键字匹配查询方式获取的演化信息会比较粗略。

(2)使用 SPARQL Query 查询可以进一步提炼查询结果，返回构件、版本信息等相关信息。例如运行如下查询语句可以得到 $SA_1\_B_1$ 的构件版本集合为{$C_1\_V_{1.0}$，$C_3\_V_{1.0}$，$C_5\_V_{1.0}$，$C_6\_V_{1.0}$，$C_8\_V_{1.0}$}。

```
PREFIX cbso:〈http://www.co-ode.org/ontologies/ont.owl#〉
    SELECT ?version
    WHERE
    {
      ?version cbso:Part-of ?baseline.
      ?baseline cbso:Version-of ?configuration.
      FILTER regex (?baseline," SA1_B1" ) //过滤检索
      }
```

(3)基于描述逻辑的 DL Query 查询方式，可以结合析取(or)、合取(and)、非(not)逻辑和值约束(value)、only 等获取关于特定的概念、属性(对象属性、数据属性)或者个体的信息。如要查询 6 月份到 7 月份的版本，则可以描述成 Version and (timeValue value 20130601) and (timeValue value 20130630)。

4.1.2 基于推理机的蕴含演化信息获取

本体模型通常与推理机相结合，以推导出本体模型中蕴含的知识。其中，Jena 推理机[15]是一种典型的、基于规则的推理机系统。为了更深入地理解软件的演化性，本节讨论基于 Jena 推理机实现对构件化软件蕴含演化信息推理获取。

• 规则的定义：除 Jena 推理机中定义的通用规则外，针对构件化的演化信息的特殊需要，本文定义了一组扩展规则作为通用规则的补充。

(1)演化信息的传递性规则

传递性规则表达的含义：若 a 的变化引发 b 的变化，b 的变化引发 c 的变化。那么，a 可引发 c 的变化。查询规则定义如下：

```
[rule1:(?a owl:Part-of ?b)(?b owl:Part-of ?c)->(?a owl:Part-of ?
```

c)]//Part-of 传递性

[rule2:(?a owl:Prior-order ?b)(?b owl:Prior-order ?c)->(?a owl:Prior-order ?c)]

[rule3:(?a owl:Next-order ?b)(?b owl:Next-order ?c)->(?a owl:Next-order ?c)] //rule2 和 3 版本序列推理

(2)传递性规则的扩展规则

传递性规则的扩展规则表达的含义：如果 a 与 c 之间关系具有传递性，c 与 b 为 Version-of 关系，那么 a 与 b 也具有 Version-of 关系。利用这种关系可以获得构件的版本集合、配置的基线集合和基线中的构件版本集合。规则定义如下：

[rule4:(?b owl:Instance-of 'Component')(?a owl:Prior-order ?c)(?c owl:Version-of ?b) ->(?a owl:Version-of ?b)]// 查找构件的版本集合规则

[rule5:(?b owl:Instance-of 'Configuration')(?a owl:Prior-order ?c)(?c owl:Version-of ?b) ->(?a owl:Version-of ?b)]// 查找配置的基线集合规则

[rule6:(?a owl:Version-of ?b)(?b owl:Instance-of 'Configuration')(?c owl:Version-of ?d)(?d owl:Instance-of 'Component')(?c owl:Part-of ?a) ->(?a owl:belongTo ?c)]//查找基线中构件版本集合规则

(3)辅助推理规则

辅助推理规则主要用于对某些演化信息细节的推理，如某人参与的版本集合、配置集合的查询等。例如如下规则：

[rule7:(?a owl:Work-of ?b) ('Component' owl:Part-of ?b)(?c owl:Instance-of 'Component') -> (?a owl:Develop-of ?c)]//某人负责开发的构件集

[rule8:(?a owl:Work-of ?b) ('Configuration' owl:Part-of ?b)(?c owl:Instance-of 'Configuration') -> (?a owl:Develop-of ?c)]// 某人负责开发的配置集

· 本体推理的实现：把以上几条规则写入 Jena 推理规则库中，然后基于相关的 OWL 文档进行推理，部分实现代码如下：

```
OntModel mrdf = ModelFactory. createOntologyModel (OntModelSpec. OWL_MEM,null );//创建 RDF 模型
Model schema = ModelLoader. loadModel ( " file: testing/reasoners/bugs/Component Software evolution. owl");//读入框架模型
List Com_rule=Rule. rulesFromURL("file:f:/Query. rules");
//把所需规则加入到 Jena 推理规则库
String queryString="……";
//应用本体查询语言 SPARQL 构件查询语句
Reasoner Com _ reasoner = new GenericRuleReasoner (Rule. parseRules (Com_rule));//创建推理机
Com_reasoner=Com_reasoner. bindSchema(schema);
//绑定框架模型和推理机
InfModel Com_data = ModelFactory. createInfModel(Com_reasoner,mrdf);
//创建包含推理关系的数据模型
Query query=QueryFactory. create(queryString);
//创建查询对象
QueryExecution Qu _ exe = QueryExecutionFactory. create (queryString,Com_data);//执行查询
ResultSet results=(ResultSet) Qu_exe. execSelect();
//存储结果
```

根据以上程序可得到构件版本集合或者系统基线集合，这些集合按照版本号递增（版本不一定连续）的顺序进行排序。例如查询李四负责开发的构件版本，根据规则 7 推理首先得到人员与构件的开发关系（Develop-of）。然后通过本体查询语言 SPARQL 查询语句得到人员'李四'负责开发的构件集合。进而利用规则 4 得到构件 C_1 版本集合为 $SC1=\{C_1\_V_{1.0}, C_1\_V_{1.1}, C_1\_V_{1.2}, C_1\_V_{2.0}, C_1\_V_{2.1}, \cdots\}$；构件 C_3 的版本集合为 $SC3=\{C_3\_V_{1.0}, C_3\_V_{1.1}, C_3\_V_{1.2}, \cdots\}$，其他构件 C_5、C_6、C_8 类似。利用规则 5 和规则 6 查询得到系统基线集合为：$\{\{C_1\_V_{1.0}, C_3\_V_{1.0}, C_5\_V_{1.0}, C_6\_V_{1.0}, C_8\_V_{1.0}\}, \{C_1\_V_{2.0}, C_3\_V_{3.0}, C_5\_V_{4.0}, C_6\_V_{5.0}, C_8\_V_{3.0}\}, \{C_1\_V_{4.0}, C_3\_V_{4.0}, C_5\_V_{5.0}, C_6\_V_{7.0}, C_8\_V_{5.0}\}, \{C_1\_V_{8.0}, C_3\_V_{6.0}, C_5\_V_{6.0}, C_6\_V_{8.0}, C_8\_V_{7.0}\}, \{C_1\_V_{9.0}, C_3\_V_{7.0}, C_5\_V_{9.0}, C_6\_V_{9.0}, C_8\_V_{9.0}\}\}$，此集合可以作为后续软件演化信息度量的输入。

4.2 构件化软件演化信息度量

基于前述基于规则推理得到的构件版本集合或者系统基线集合，可以定义相应的软件演化信息度量公式。正如第 3 节所论述的"构件或者构件版本表现为一组与一个逻辑概念密切相关的文件集合"，在这里构件的变化可以认为是文件数目的变化。本节首先定义函数 $Num\_Files(S,i)$ 用于计算某个构件版本的文件数目，其中 S 为版本号递增（不一定连续）的有序构件版本集合，i 为序列号（例如前面版本集合 $SC3$ 中版本 $C_3\_V_{1.1}$ 所对应的序列号为 2）。基于函数 Num_Files，我们定义了一组演化信息度量公式用于度量构件或者系统的变化程度，包括用于度量构件变化程度的 C_EP，SC_EP，C_EEP 和 C_LEP 公式，以及度量系统变化程度的 B_EP，B_EEP 和 B_LEP 公式，具体如下。

EP：(Evolution of Property)用于计算软件属性的演化，在构件化软件演化过程中，属性的演化可能发生在两个层次，即构件层次和系统层次。在构件层次，对应有 $C\_EP(C,i)$ 和 $SC\_EP(C,i)$；在系统层次则对应有 $B\_EP(B,i)$。其中 C 表示某个构件，而 i 代表经过推导得到的构件 C 所对应版本集合中某个版本的序列号；同理，B 代表系统或者软件体系结构，i 代表其对于某个基线的序列号。

(1)针对特定构件版本，EP 表示某个构件版本与前一个相邻版本间的属性变化，具体表现为两个构件版本间文件数目的变化（在如 4.1.2 节获取到的版本集合中构件 C_1 的首个版本的 $C\_EP(C_1,1)$ 为该版本的文件数目）。如式(1)所示：

$$C\_EP(C,i)=\begin{cases} Num\_Files(C,i), & i=1 \\ |(Num\_Files(C,i)-Num\_Files(C,i-1)|, & i>1 \end{cases} \tag{1}$$

C_EP 反映某个构件属性的变化，为了统计构件生成最新版本前所有的版本属性变化的总和，定义式(2)，m 为常数，表示某个构件中版本的个数。

$$SC\_EP(C)=\sum_{i=1}^{m} C\_EP(C,i), i\geqslant 1 \tag{2}$$

(2)针对构件化软件系统的演化，可以从纵向和横向两个角度加以考量。从纵向来看，系统的整体结构未发生变化（即系统的软件体系结构未发生变化），但组成系统的构件发生了变化。此时，系统演化可以理解为文件集合的变化，表示某个系统基线中各构件版本 C_EP 值的总变化，即该基线的各构件版本的 C_EP 值的总和。如式(3)所示，其中映射函数 $map(B,i,j)$ 是返回序列号为 i 的基线中序列号为 j 的构件，n 为常数，表示基线中构件的个数。

$$B\_EP(B,i)=\sum_{j=1}^{n}C\_EP(map(B,i,j),i),i\geqslant 1 \quad (3)$$

从横向来看，构件化软件的演化则主要体现在软件结构上的差异性，为此，文献[17]中的编辑距离，将软件结构之间的差异性定义为图之间变换所需要编辑操作（插入、剔除、替换）的最小数目，定义如下：

$$B\_EP(B,i)=|NumOF(InsertEdge(B,i,i-1))+NumOFDeleteEdge((B,i,i-1))+NumOFInsertNode(B,i,i-1)))+NumOFDeleteNode(B,i,i-1)))+NumOFReplaceNode(B,i,i-1)))| \quad (4)$$

LEP:(Latest Evolution of Property)该度量主要衡量软件的最新变化。针对构件化软件，LEP 可以在构件和系统两个层面度量，分别对应 $C\_LEP(C,i,j)$ 及 $B\_LEP(B,i,j)$，其中，i,j 为构件版本序列号或者系统基线序列号，C 为构件名，B 为系统。

$C_LEP(C,i,j)$作为某个构件演化的整体性指标，不仅关注构件属性的总变化，而且关注最新构件版本的变化，对每个构件版本的 C_EP 增加权重函数 $2^{i-maxRank}$（其中 $maxRank$ 为参与度量最新构件版本对应的版本序列号），减弱过去版本属性变化的重要性，统计构件生成最新版本前所有的版本属性变化的加权总和，即构件 C 从第 i 个构件版本到第 j 个构件版本的 C_EP 加权求和。如式(5)所示，其中 $i,j,maxRank$ 均为版本序列号。

$$C\_LEP(C,i,j)=\sum_{k=i}^{j}C\_EP(C,k)*2^{k-j},1\leqslant i\leqslant k\leqslant j\leqslant maxRank \quad (5)$$

$B_LEP(B,i,j)$作为系统演化的整体性指标，反映的是系统总的变化。因此，该度量从系统全局的角度来分析，重点关注最新基线的变化，对每个基线的 B_EP 增加权重函数 $2^{i-maxRank}$（其中 $maxRank$ 为参与度量最新基线版本对应的基线序列号），统计基线生成最新版本前所有的基线属性变化的加权总和，即系统 B 从第 i 个基线到第 j 个基线的 B_EP 加权求和。

$$B\_LEP(B,i,j)=\sum_{k=i}^{j}B\_EP(B,k)*2^{k-j},1\leqslant i\leqslant k\leqslant j\leqslant maxRank \quad (6)$$

EEP:(Earliest Evolution of Property)该度量关注属性的早期变化，同样，EEP 也有构件和系统两个层面的度量，分别对应 $C_EEP(C,i,j)$和 $B\_EEP(B,i,j)$。

$C_EEP(C,i,j)$不仅反映构件属性的总变化，而且强调早期版本的变化。对每个构件版本的 C_EP 增加权重函数 $2^{minRank-i}$（其中 $minRank$ 为参与度量早期旧构件版本对应的版本序列号），减弱新版本属性变化的重要性，即构件 C 从第 i 个构件版本到第 j 个构件版本的 C_EP 加权求和。

$$C\_EEP(C,i,j)=\sum_{k=i}^{j}C\_EP(C,k)*2^{i-k},minRank\leqslant i\leqslant k\leqslant j \quad (7)$$

类似地，$B_EEP(B,i,j)$从系统的角度分析系统从初始版本到最新版本所有的属性变化的加权总和，即系统 B 从第 i 个基线到第 j 个基线的 B_EP 加权求和。

$$B\_EEP(B,i,j)=\sum_{k=i}^{j}B\_EP(B,k)*2^{i-k},minRank\leqslant i\leqslant k\leqslant j \quad (8)$$

图 2 展示了 4.1.2 节中得到的构件版本集合的演化历史。观察图 2 中 C_LEP、C_EEP、B_LEP 和 B_EEP 等值的变化，可以比较构件（系统）的变化程度和变化的稳定性。

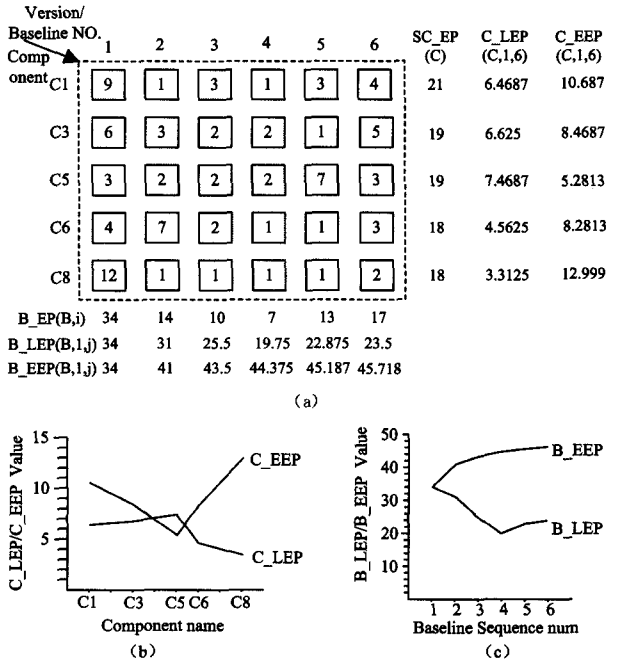

图 2 构件化软件演化信息度量

从图 2(a)横向来看，该段演化历史包含有 5 个构件，每个构件对应有 6 个版本，方框内的数值表示该构件版本的 C_EP 值。例如，从图 2(b)中可以看出：① 构件 C_3 和构件 C_5 的 SC_EP 值相同，但 C_3 的 C_LEP 值(6.625)比 C_5 的 C_LEP 值(7.4687)小，说明 C_3 的变化程度在近期要小于 C_5 的变化程度；同理，C_3 的 C_EEP 值(8.4687)比 C_5 的 C_EEP 值(5.2813)大，则说明 C_3 的变化程度在早期大于 C_5 的变化程度。② 构件 C_8 的 C_EEP 值(12.999)大于其 C_LEP 值(3.3125)，说明 C_8 在早期的变化程度大于其在近期的变化程度。

从图 2(a)纵向来看，该段演化历史为 6 个系统基线（局部），其中 $B_EP(B,i)$中的 i 表示基线序列号，$B_LEP(B,1,j)$计算的是该系统从第 1 个基线到第 j 个基线之间的 B_LEP 值，B_EEP 类似。观察图 2(c)中 B_LEP 与 B_EEP 的变化，可知①B_LEP 值从基线 1 到 4 时递减，随后又在基线 5 和基线 6 增大，说明系统的近期变化程度经历了逐渐减小而随后上升的过程。②各基线的 B_EEP 逐渐增大，但基线在 3 到 6 时 B_EEP 之间的差值区分不大（分别为 0.875，0.812，0.571），说明从宏观上看，系统的早期变化程度逐渐趋于稳定。

5 系统原型

图 3 给出了基于本体的构件化软件演化信息获取及度量平台的系统原型。整个系统主要由本体抽取器、基于 Jena 推理机的查询器以及演化信息度量模块 3 个部分构成。其中，本体抽取器负责从基于构件的软件配置管理系统 JBCM 中半自动地生成某个具体的构件化软件系统的演化本体模型；基于 Jena 推理机的查询器则按照指定的规则查询直接的或者蕴含的软件演化信息，例如符合某个查询条件的构件版本集合或者系统基线集合；演化信息度量模块负责计算各类 EP、LEP 和 EEP。

（下转第 231 页）

[8] 王品,黄广君. 信息检索中句子相似度计算[J]. 计算机工程,2011,37(12):38-40
[9] 李茹,王智强,等. 基于框架语义分析的汉语句子相似度计算[J]. 计算机研究与发展,2013,50(8):1728-1736
[10] 孙茂松. 基于互联网自然标注资源的自然语言处理[J]. 中文信息学报,2011,25(6):26-32
[11] 百度 Q-T 语义一致大赛 [DB/OL]. http://openresearch.baidu.com/activityindex.jhtml? channelId=452
[12] NLPIR 汉语分词系统[DB/OL]. http://ictclas.nlpir.org/
[13] 董振东,董强. 知网[DB/OL]. http://www.keenage.com
[14] 梅家驹,竺一鸣,高蕴琦,等. 同义词词林[M]. 上海:上海辞书出版社,1993:106-108

(上接第 200 页)

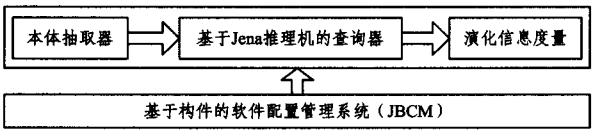

图 3 构件化软件演化信息获取及度量系统原型

结束语 在构件化软件开发中,利用演化信息能够更好地辅助软件的开发。传统的软件演化信息通常以文件或者项目作为软件变化的基本单元,不能有效地支持构件化软件演化信息的存储和检索。为此,本文在基于构件的软件配置管理模型的基础上,对构件化软件演化信息进行本体建模,借助描述逻辑 DL Query 以及基于属性相关性的 SPARQL 查询等机制实现直接演化信息查询;同时,通过定义规则和在 Jena 推理机中实现构件化软件蕴含演化信息的获取;最后提出了一种预测演化趋势的构件化软件演化度量的方法,为后期的构件化软件维护工作提供了基础。

需要指出的是本文的工作与 Christoph Kiefer 的基于本体的软件资产库挖掘[13]相似,不过他们的研究面向对象软件的演化,侧重点是扩展 SPARQL 语言以支持基于相似度的查询,以及通过简单的示例说明在软件演化可视化、度量、本体推理上的应用;而本文的工作针对的是构件化软件演化,着重于基于本体和 Jena 推理机的构件化软件演化信息获取及演化度量。在未来的工作中,将完善目前的系统原型,并将其封装成 Web-Service 的形式。

参 考 文 献

[1] Lehman M, Belady L. Program Evolution: Processes of Software Change [M]. London Academic Press: London, 1985: 538-540
[2] Gîrba T, Ducasse S. Modeling History to Analyze Software [J]. Journal of software maintenance and evolution: research and practice, 2006, 18: 207-236
[3] Gîrba T. Modeling History to Understand Software Evolution [D]. Fakultät der Universität: Berne, 2005: 13-19
[4] Morse T. CVS[J]. Linux Journal, 1996(21es): 3, 1996
[5] Subversion[OL]. [2013-12-15]. http://subversion.tigris.org/
[6] Robbes R, Lanza M. Versioning systems for evolution research [C]// 8th International Workshop on Principles of Software Evolution, 2005 (IWPSE 2005). IEEE Computer Society, 2005: 155-164
[7] Chu-Carroll M C, Wright J, Shields D. Supporting aggregation in fine grained software configuration management [C] // Formal Software Engineering FSE'02. ACM Press, 2002: 99-108
[8] Dig D, Manzoor K, Johnson R, et al. Refactoring-Aware Configuration Management for Object-Oriented Program [C] // the 29th International Conference on Software Engineering (ICSE'07). 2007: 427-436
[9] Robbes R. Mining a Change-Based Software Repository [C] // Proceedings of the Fourth International Workshop on Mining Software Repositories (MSR '07). 2007: 15-22
[10] Xing Z, Stroulia E. Refactoring detection based on umldiff change-facts queries [C] // Proc. WCRE'06. 2006: 263-274
[11] Dig D, Comertoglu C, Marinov D, et al. Automatic detection of refactorings in evolving components [C] // Proc. ECOOP'06. 2006: 404-428
[12] Taneja K, Dig D, Xie Tao. Automated detection of api refactorings in libraries [C] // ASE'07. ACM, 2007: 377-380
[13] Kiefer C, Bernstein A. Mining Software Repositories with iSPARQL and a Software Evolution Ontology [C] // Fourth International Workshop on Mining Software Repositories (MSR'07). 2007
[14] Matthew H. A practical Guide to Building OWL Ontology Using the Protégé-OWL Plugin and Code Tools [Z]. [2013-05-10]
[15] Restol, Jena2. A semantic Web Framework [OL]. [2013-12-20]. http://Jena.Sourceforge.net
[16] OWL Web Ontology Language Overview [OL]. [2013-03-19]. http://www.w3.org/TRlowl-features/
[17] Sager T, Bernstein A, Pinzger M, et al. Detecting Similar Java Classes Using Tree Algorithms [C] // Proc. of the 2006 Int. Ws. on Mining Software Repositories (MRS '06). New York, NY, 2006
[18] Stevens R, De Roover C, Noguera C, et al. A History Querying Tool and its Application to Detect Multi-version Refactorings [C] // 17th European Conference on Software Maintenance and Reengineering. 2013: 335-338
[19] Prete K, Rachatasumrit N, Sudan N, et al. Template-based reconstruction of complex refactorings [C] // Proc. of the 2010 IEEE Int. Conf. on Software Maintenance. 2010: 1-10
[20] 曹居易,石玲. 基于 OWL 的软件工程数据建模[J]. 计算机研究与发展,2009,46(增刊):214-221
[21] 李季,刘春梅. 基于本体的可信软件演化框架模型[J]. 计算机应用研究,2010,27(12):4551-4554
[22] 何文民,沈国华,黄志球. 基于本体的特征模型演化的一致性验证[J]. 计算机应用研究,2013,30(7):2072-2076
[23] 张路,谢冰,梅宏,等. 基于构件的软件配置管理技术研究[J]. 电子学报,2001,29(2):266-268
[24] 钟林辉,谢冰,邵维忠. 扩充 CDL 支持基于构件的系统组装与演化[J]. 计算机研究与发展,2002,39(10):1361-1365