通用的软件产品线领域与应用特征模型演化同步框架

黄 洋 沈立炜 彭 鑫
(复旦大学计算机科学技术学院 上海 201203)

摘 要 软件产品线领域特征模型和应用特征模型都会发生独立的演化,在产品线的整体演化过程中必须使其始终保持一致,然而,分别为基于不同描述方式的特征模型创建同步设施往往费时、容易出错。因此,提出一个通用的软件产品线领域与应用特征模型的演化同步框架,该框架提供一套统一的特征模型元模型描述方式,并且基于该元模型提出了演化同步规则。不同的软件产品线开发组织使用该框架时仅需定义特定的特征描述与通用描述方式之间的转换。最后,通过一个实例对框架的可用性进行了验证。

关键词 产品线,特征模型演化,同步
中图法分类号 TP311.5 **文献标识码** A

Generic Framework for Synchronizing Domain Feature Model and Application Feature Models in Software Product Line Evolution

HUANG Yang SHEN Li-wei PENG Xin
(Department of Computer Science and Technology, Fudan University, Shanghai 201203, China)

Abstract Software Product Line domain feature model and application feature models need to keep consistency during evolution, however they usually evolve separately. Building synchronization facility for each kind of feature model is time-consuming and error-prone. So we proposed a generic framework for synchronizing domain feature model and application feature models during evolution, including a generic feature meta-model and the synchronization rules based on the meta-model. Using the framework, different software organizations just need to define the transformation between their specific feature model and the generic feature model. In addition, we used an example to verify the usability of the framework.

Keywords Software product line, Feature model evolution, Synchronization

1 引言

软件产品线是共享一组受控的公共特征,满足特定市场需求,并且在一组预定义的公共核心资产基础上开发而成的一系列软件系统[1]。它主要包括领域工程与应用工程这两个阶段,前者关注于创建可复用的领域核心资产,而后者则基于这些资产开发特定的应用产品。

当前,基于特征的软件产品线开发方法已经成为主流。其中,特征模型是产品线方法的核心,它通过图形化与格式化的手段描述领域需求,并将特征作为与最终用户交流的一阶实体[2,3]。特征模型进一步分为领域特征模型与应用特征模型,在领域工程阶段所创建的领域特征模型着重描述领域应用间的共性与可变性,而它将在应用工程中被定制成为不含有可变性的应用特征模型[1-3]。另外,特征模型并没有一个统一的图形化与形式化标准,因此许多研究者与软件公司都提出了自己的特征模型,包括对特征及不同建模元素的定义。

另一方面,软件产品线处在不停的演化过程中。由于需求变更、功能升级或技术革新等因素,不论是领域特征模型还是应用特征模型都将进行演化,例如加入或修改特征等,同时它们的演化可能是相互独立的。长此以往,领域与应用特征模型之间将产生不一致的情况,即应用需求与领域需求之间不再同步,这种不一致将对软件产品线的维护产生严重影响,并且使得开发组织对整个软件产品线失去控制[4]。

因此,领域特征模型与应用特征模型之间的演化同步成为成功实施软件产品线开发方法的一个重要因素。通过自动化的手段或工具识别模型演化情况,随后基于同步规则对领域或应用特征模型进行相应修改,是达到该目标的有效方法。然而,特征模型种类众多,采用该方法需要为各种特征模型单独建立同步设施,这是一个冗余的、复杂的且容易出错的工作。

领域特征模型与应用特征模型的演化同步问题已有相关研究[5,6]。文献[5]从模型同步的角度对该问题进行了分析,并提出了一系列模型转换规则来实现领域特征模型和应用特征模型之间的同步。文献[6]定义了两种操作即 create_con-

到稿日期:2013-01-13 返修日期:2013-06-15 本文受教育部博士点基金(20100071110031)资助。
黄 洋(1988—),男,硕士生,主要研究方向为软件产品线,E-mail:hapmile@163.com;沈立炜(1982—),男,博士,讲师,主要研究方向为软件复用、软件产品线;彭 鑫(1979—),男,博士,副教授,CCF高级会员,主要研究方向为软件产品线、软件体系结构、软件再工程等。

figuration 和 differencs 来实现领域到应用的传播。然而文献[5,6]都只考虑了领域特征模型演化向应用特征模型传播的情况，而没有考虑应用特征模型演化的情况，而且两种方法都是针对一种特定的特征模型而提出的。

针对以上情况，本文提出了一个通用的软件产品线领域特征模型与应用特征模型的演化同步框架。该框架包含一套统一的、通用的特征模型描述方式(元模型)，该描述能够与符合规范特征语义的任意特征模型进行互相转换。同时，在该描述的基础上提出通用的演化同步规则。因此，采用不同种类特征模型的产品线组织都能够使用该框架，共享其中的同步规则部分，而仅需指定特定的特征描述与通用描述方式之间的转换。

本文第 2 节将介绍软件产品线中特征模型的演化场景及其相应的同步需求；第 3 节对同步框架进行具体描述，包括演化同步的规则；第 4 节将在概览不同特征模型的基础上提出通用的特征模型元模型及其描述方式；第 5 节是一个使用框架的软件产品线同步实例；最后对本文进行总结。

2 产品线的特征模型演化

2.1 演化场景

产品线的领域特征模型和应用特征模型都会各自发生演化，我们将演化场景分为以下 3 种类型：

1)只有领域特征模型发生演化。此时需要将领域特征模型的变化传播给每个应用特征模型。应用工程中的特征模型定制生成实际就是这种情况的一个特例，此时还没有应用特征模型，通过对领域特征模型所有可变性的定制，得到最初的应用特征模型。

2)只有应用特征模型发生演化。此时首先要将应用特征模型的变化反馈给领域特征模型，如果领域特征模型因此也发生了变化，则需要再向所有应用进行传播。

3)领域特征模型和应用特征模型都发生了演化。此时首先需要将应用模型的变化反馈给领域特征模型，然后领域特征模型再将变化(包括自身演化的变化和反馈产生的变化)传播给所有应用特征模型。

由上述分析可以总结出，产品线领域和应用特征模型同步的步骤为首先将应用特征模型的变化反馈给领域特征模型，然后将领域特征模型的变化传播给所有应用特征模型，该过程见图 1。

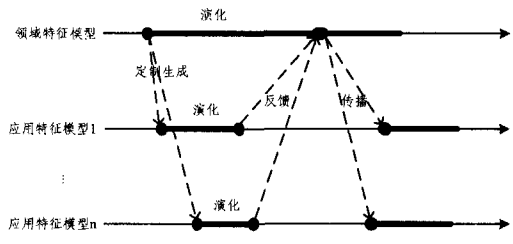

图 1 产品线特征模型演化场景

3 特征模型演化同步框架

图 2 为特征模型演化同步框架的示意图，下面详细介绍框架实施的具体步骤。

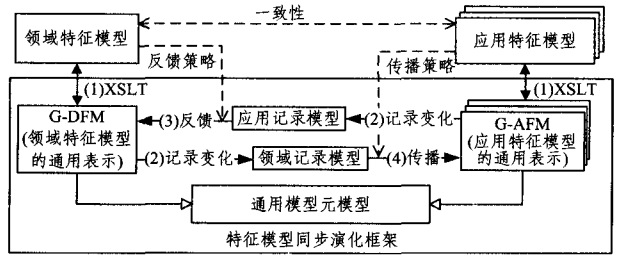

图 2 特征模型演化同步框架

3.1 特定特征模型与通用特征模型的转换

首先将领域特征模型和应用特征模型分别转换为 G-DFM(领域特征模型的通用描述)和 G-AFM(应用特征模型的通用描述)，以便在通用描述的基础上实施同步；在同步完成后，再将 G-DFM 和 G-AFM 转换为特定的特征模型。特征模型的通用描述以及与特定模型之间的转换在第 4 节详述。

3.2 识别记录特征的变化类型

我们定义了一个记录模型，用来记录特征模型本次同步相对于上次同步后发生的变化。领域特征模型和每个应用特征模型都有一个相应的记录模型。每个记录模型包含了相应特征模型中每个特征的变化情况。对比当前特征模型和上次同步后特征模型的每个特征，发现特征被改变后，由用户辅助标注特征的变化类型以及变化的原因，然后存储在记录模型中。可能的变化类型标注见表 1，其中特征的可变性被改变只会发生在领域特征模型对比识别中，因为应用特征模型中不包含可变性。

表 1 变化类型的识别

对比识别的变化	可能标注的变化类型
特征被修改	特征被修改
特征被删除	特征被删除 被新添加的特征替换
新添加的特征	新添加的特征 替换原来的某个特征
特征的可变性被改变(领域)	特征的可变性被改变

3.3 反馈

由 2.1 节分析可知，同步应先进行反馈。反馈过程是依次对每个应用特征模型执行反馈。遍历该应用特征模型的记录模型，对发生变化的特征进行反馈，具体的反馈规则见表 2。第一列是应用特征模型中可能发生的特征变化类型，包括特征被修改或替换、特征被删除和添加一个新特征。第二列是该特征在领域中的可变性类型，以及根据不同变化类型和可变性类型领域可能采取的反馈操作。其中特征修改和特征替换都是指特征因应用的需求而发生了改变，或进行了修改，或直接用新的特征予以替换，它们的反馈规则是一样的，因而被归为一类。领域工程师根据特征变化的原因(如错误修正、功能改动等)和领域知识，可能会采取 3 种方式应对反馈：1)接受，若认为该特征变化是一种修正或者完善，且应该被推广到所有应用中，则接受；2)保持不变，若认为该特征变化是该应用的特定需求，则保持不变；3)吸纳，若认为该特征变化表示了领域中某个新的方面(如新的功能、新的解决方案等)，应该被吸纳到领域中，则将该特征变化作为可变性添加到领域中。根据特征变化类型和特征可变性类型不同，以上 3 种方式的具体操作会略有不同。同时，反馈导致领域特征模型发生的变化也要记录在领域特征模型的记录模型中。

表 2 反馈规则

应用特征模型的变化		领域特征模型相应的规则
变化类型	可变性类型	操作
特征被修改或者替换	必选	接受。
		保持不变。
		新建一个必选的特征组替代原来的必选特征，选择基数为[1..1]，包含两个变体，一个是原来的必选特征，一个是应用中变化的特征。
	可选	接受。
		保持不变。
		新建一个可选的特征组替代原来的可选特征，选择基数为[1..1]，包含两个变体，一个是原来的可选特征，一个是应用中变化的特征。
	变体	接受。
		保持不变。
		将应用中变化的特征作为一个新的变体添加到父特征组中。
特征被删除	必选	将必选特征修改为可选。
		保持不变。
		从领域中删除该必选特征。
	可选	保持不变。
		从领域中删除该可选特征。
	变体	保持不变。
		从领域中删除该变体，在需要时调整父特征组的选择基数。
添加一个新特征	父特征不是特征组	保持不变。
		将该特征作为必选特征添加到领域中。
		将该特征作为可选特征添加到领域中。
	父特征是特征组	保持不变。
		将该特征作为变体添加到父特征组中，在需要时调整父特征组的选择基数。

3.4 传播

反馈之后，进行传播。对每个应用各自进行传播，遍历领域记录模型，对发生变化的特征进行传播，传播规则见表 3。第一列是领域特征模型中可能发生的特征变化类型，包括特征被修改或替换、特征被删除、特征可变性被改变和添加一个新特征。此时，应用工程师根据特征的变化类型和应用需求，可以采取两种方式应对传播：1)接受，若认为该特征变化适用于该应用，则接受，并采取相应操作；2)保持不变，若认为该特征变化不适用于该应用，则保持不变。至此，领域特征模型和应用特征模型完成了同步，重新恢复了一致性。

表 3 传播规则

领域特征模型的变化	应用特征模型相应的规则	
变化类型	可变性类型	动作
特征被修改或者替换	—	接受
		保持不变
特征被删除	—	接受
		保持不变
可变性被改变	—	重新定制
		保持不变
添加一个新特征	—	定制
		保持不变

4 通用特征模型

4.1 特征模型

特征模型是一个分层次的特征树，每一个父特征和它的子特征的关系包括[7]：

1)必选：父特征被选择，子特征必须选择。

2)可选：子特征是可选的。

3)多选一：有且只有一个子特征被选择。

4)多选多：至少一个子特征被选择。

除了父特征和子特征的关系，特征模型中还存在两种跨树的约束：

1)依赖：f1 依赖 f2 表示，选择 f1 必须也要选择 f2。

2)互斥：f1 与 f2 互斥表示，f1 和 f2 不能同时选择。

以上是基本的特征模型，此外还有一些扩展的特征模型，如基于基数的特征模型[8]和包含属性的特征模型[3,9]。除此以外，不同的特征模型工具也有自己的描述方法。而实际上这些描述方法的语义都是相同的，特征之间的关系都遵从上面所说的关系，只是描述方法不同，且每种描述中的特征都有自己特定的属性。

4.2 通用特征模型

为了能在不同的特征模型上进行一致性的演化，本文提出了一种通用的特征模型描述方法。由于不同特征模型底层都是基于 XML 保存的，因此通过记录 XML 的标签(tag)，就能够完整地保存原来模型的信息；同时识别出其中的特征及可变性信息，予以记录，父特征和子特征的关系通过将子特征嵌套在父特征中来记录。用识别的特征可变性信息支持领域和应用间特征模型的演化同步。通用特征模型的元模型见图 3。元模型中的基类是 Tag，即用来存储 XML 文件中的标签，而特征模型以及特征都是特殊的标签，Tag 类包含一个 tagName 属性，用来保存标签的名称，Tag 又可以包含 subTags 和 Property。类 Property 用来记录标签的属性信息，name 存储属性的名称，value 存储属性的值。FeatureModel 类描述整个特征模型，它继承 Tag 类。Feature 类继承 Tag 类，用来描述特征，featureName 存储特征的名称，该名称也作为特征的唯一标示，子特征会嵌套聚集在父特征中(通过 subTags)；variableType 存储特征的可变性，variableType 的类型为枚举类型 VariableType，可选值为 mandatory(代表该特征是父特征的必选子特征)、optional(代表该特征是父特征的可选子特征)和 variant(代表该特征的父特征是多选一或多选多特征组，自己为其中一个变体)；variableGroup 标示特征是否是一个特征组，即它与它的子特征之间是否是多选一或者多选多的关系；cardinality 的类型是 Cardinality，用来存储这个特征组子特征的选择基数，Cardinality 类包含 min 和 max 两个属性，存储特征组子特征可选的最小数量和最大数量，如对于多选一特征组，min 和 max 的取值都为 1；此外特征还有两种引用 requires 和 excludes，用来存储该特征依赖和互斥的特征。

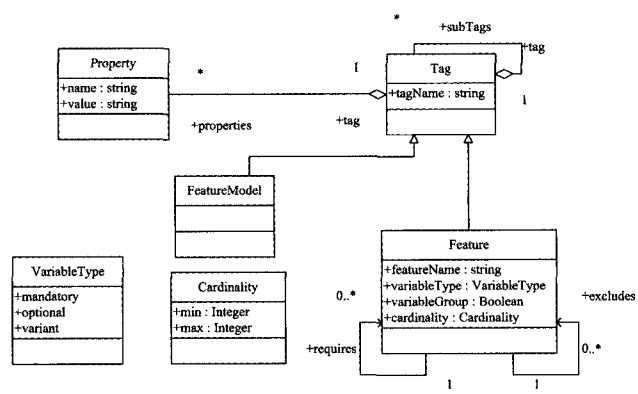

图 3 通用特征模型元模型

4.3 XSLT 转换

XSLT 是 W3C 提出的一种对 XML 文档进行转化的语言[10]。本文使用 XSLT 来实现特定特征模型和通用特征模型的转换。如对于图 4 中的特征模型，表 4 列出了它在 FeatureIDE 中的描述，以及将 FeatureIDE 描述转换为通用描述的 XSLT 代码节选和转换结果。

其中特征的父子关系和可变性信息被抽取保存为：父特征 Smart_Home 嵌套两个子特征 Heating_Management 和 Alarm，Heating_Management 是必选的（variableType="mandatory"），Alarm 是可选的（variableType="optional"），而特征模型的其它信息，如标签 featureModel 的 chosenLayoutAlgorithm 属性通过 property 来保存，而标签 featureOrder 通过 tag 予以记录保存。由于通用特征模型保存了原特征模型的所有信息，包括特征可变性信息以及其它特定信息，因此可以在之后通过 XSLT 再将通用特征模型转化为原来的特征模型。

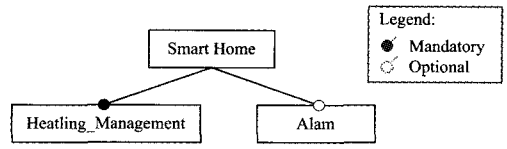

图 4 特征模型示例

表 4 FeatureIDE 的描述、XSLT 转换代码和转换结果

示例特征模型在 FeatureIDE 中的描述	⟨featureModel chosenLayoutAlgorithm="1"⟩ ⟨struct⟩ ⟨and mandatory="true" name="Smart_Home"⟩ ⟨feature mandatory="true" name="Heating_Management"/⟩ ⟨feature name="Alarm"/⟩ ⟨/and⟩ ⟨/struct⟩ ⟨constraints/⟩ ⟨comments/⟩ ⟨featureOrder userDefined="false"/⟩ ⟨/featureModel⟩
FeatureIDE 描述转换为通用描述的 XSLT 代码	⟨xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"⟩ ⟨xsl:template match="featureModel"⟩ …… ⟨! —— 转换特征——⟩ ⟨xsl:template match="feature"⟩ ⟨! —— 读取特征的可变性 ——⟩ ⟨xsl:variable name="variableType"⟩ ⟨xsl:if test='(@mandatory)="true"'⟩mandatory⟨/xsl:if⟩ ⟨xsl:if test='not((@mandatory)="true")'⟩optional⟨/xsl:if⟩ ⟨/xsl:variable⟩ ⟨! ——生成通用描述的标签 ——⟩ ⟨feature tagName="feature" featureName="{@name}" variableType="{$variableType}"/⟩ ⟨/xsl:template⟩ …… ⟨/xsl:stylesheet⟩
FeatureIDE 转换后的通用模型描述	⟨featureModel⟩ ⟨property name="chosenLayoutAlgorithm" value="1"/⟩ ⟨tag tagName="struct"⟩ ⟨feature featureName="Smart_Home" variableType="mandatory"⟩ ⟨feature featureName="Heating_Management" variableType="mandatory"/⟩ ⟨feature featureName="Alarm" variableType="optional"/⟩ ⟨/feature⟩ ⟨/tag⟩ ⟨tag tagName="constraints"/⟩ ⟨tag tagName="comments"/⟩ ⟨tag tagName="featureOrder"⟩ ⟨property name="userDefined" value="false"/⟩ ⟨/tag⟩ ⟨/featureModel⟩

5 实例验证

5.1 实例描述

我们用 XSLT 和 JAVA 实现了框架原型，并用一个例子对框架进行了验证。Smart_Home 是一个经典的产品线实例，图 5 是一个 Smart_Home 产品线领域特征模型的节选及其演化情况，Smart_Home 是根特征，它的子特征包含 Heating_Management、Light_Management、UI、Security、Alarm 和 Fire_Control，其中 Alarm 是可选的，其余是必选的；另外还有一个子特征 Windows_Management（用虚线矩形框标注）是领域特征模型在演化中新添加的特征。这些特征又包含更多不同可变性的子特征，其中 Pre_defined_Value 是一个多选多的特征组，它包含 3 个变体：TV、Reading 和 Normal。此外，特征模型还包含一个特征依赖 Fire_Alarm requires Alarm。图 6 和图 7 是该产品线的两个应用及其演化情况。图 6 的应用 1 选择了所有的可选特征，以及多选多特征组 Pre_defined_Value 的全部 3 个变体；在演化中，应用 1 根据用户需求又添加了两个特征 Reomte_Control 和 Internet，其中 Reomte_Control requires Internet；另外将特征 Keypad 替换为特征 FingerPrint。应用 2 选择了部分可选特征，以及多选多特征组 Pre_defined_Value 的两个变体：Reading 和 Normal；在演化中应用 2 根据用户需求添加了特征 Ambient，并删除了特征 Fire_Sprinkler。图 8 是同步后的领域和应用特征模型。

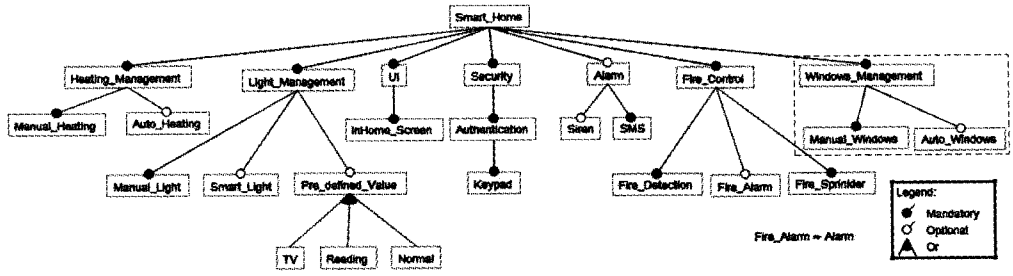

图 5　Start_Home 的领域特征模型及其演化

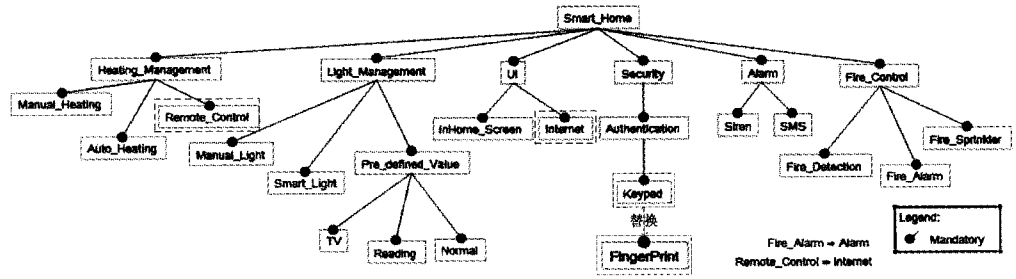

图 6　应用 1 的特征模型及其演化

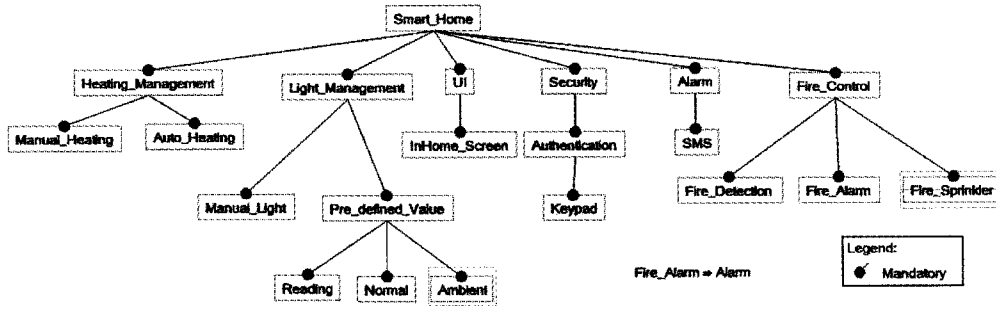

图 7　应用 2 的特征模型及其演化

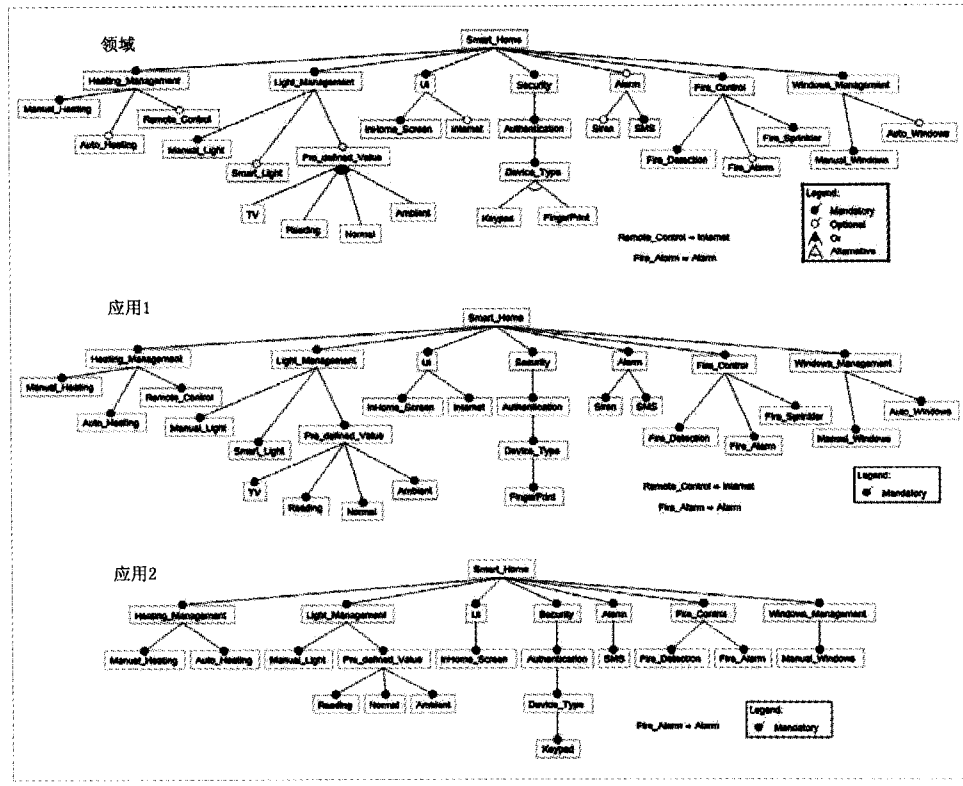

图 8　同步后的领域和应用特征模型

5.2 同步过程

1)反馈。依次对应用 1 和应用 2 执行对领域的反馈操作,反馈结果见图 8 最上面的领域特征模型。对于应用 1 中添加的两个特征:Reomte_Control 和 Internet,领域将其吸收为可选特征,并吸收特征依赖 Reomte_Control requires Internet;对于特征 Keypad 被替换为特征 FingerPrint,领域将其变为一个多选一特征组 Device_Type,包含两个变体:Keypad 和 FingerPrint。对于应用 2 添加的特征 Ambient,领域将其吸收为多选多特征组 Pre_defined_Value 的一个新的变体;而对于删除特征 Fire_Sprinkler,领域认为其是应用的一个特定需求而保持不变。

2)同步。对于应用 1,除了自身反馈引发的领域变化,领域还新添加了特征 Ambient、Windows_Management 及其子特征,应用 1 对其进行了定制,见图 8 中间应用 1 的特征模型。对于应用 2,除了自身反馈引发的领域变化,领域新添加了特征 Reomte_Control、Internet、特征 Windows_Management 及其子特征,此外特征 Keypad 变为多选一特征组 Device_Type,应用 2 对这些特征进行了定制,见图 8 最下面应用 2 的特征模型。至此,Smart_Home 的领域与应用同步完成,重新恢复了一致性。

结束语 本文对产品线领域和应用特征模型的演化场景进行了分析,提出了一个特征模型演化同步框架,其中包含针对不同种类特征模型描述提出的一种通用特征模型描述,以及领域和应用特征模型演化同步的规则。不同种类的特征模型只需提供自己与通用特征模型的 XSLT 转换代码,就可以共用文中框架的演化同步规则,实现领域和应用特征模型的一致性演化。文中 Smart_Home 的例子验证了框架的可用性。

当前的演化同步规则没有涉及到演化中可能存在的冲突问题(如多个应用对同一个特征进行了演化),接下来我们会对同步规则进行完善,以涵盖这些冲突的情况,并用大量的实例对框架进行进一步验证。

参 考 文 献

[1] Clements P C, Northrop L. Software Product Lines: Practices and Patterns[M]. Boston: Addison-Wesley, 2001

[2] Kang K C, Cohen S G, Hess J A, et al. Feature-Oriented Domain Analysis (FODA) Feasibility Study[R]. Pittsburgh: Carnegie-Mellon University Software Engineering Institute, 1990

[3] Kang K C, Jaejoon L, Donohoe P. Feature-oriented product line engineering[J]. Software, IEEE, 2002, 19(4): 58-65

[4] Peng X, Shen L, Zhao W. An Architecture-based Evolution Management Method for Software Product Line[C]// Proceedings of the 21st International Conference on Software Engineering, 2009. Boston: Knowledge Systems Institute Graduate School, 2009: 135-140

[5] Hwan C, Kim P, Czarnecki K. Synchronizing cardinality-based feature models and their specializations[C]// Proceedings of the First European conference on Model Driven Architecture-foundations and Applications, 2005. Berlin, Heidelberg: Springer-Verlag, 2005: 331-348

[6] Gamez N, Fuentes L. Software product line evolution with cardinality-based feature models[C]// Proceedings of the 12th international conference on Top productivity through software reuse, 2011. Berlin, Heidelberg: Springer-Verlag, 2011: 102-118

[7] Batory D. Feature models, grammars, and propositional formulas [C]// Proceedings of the 9th international conference on Software Product Lines, 2005. Berlin, Heidelberg: Springer-Verlag, 2005: 7-20

[8] Czarnecki K, Helsen S, Ulrich E. Staged Configuration Using Feature Models[C]// Software Product Lines, Third International Conference, SPLC, 2004. Boston, USA: Springer-Verlag, 2004: 266-283

[9] Benavides D, Trinidad P, Ruiz-Cort E S A. Automated reasoning on feature models[C]// Proceedings of the 17th international conference on Advanced Information Systems Engineering, 2005. Berlin, Heidelberg: Springer-Verlag, 2005: 491-503

[10] W3c. XSL Transformations (XSLT) Version 2.0[S/OL]. http://www.w3.org/TR/xslt20/, 2007-01-23

(上接第 151 页)

[7] Google app engine[EB/OL]. http://appengine.google.com/

[8] Microsoft azure[EB/OL]. http://www.microsoft.com/azure/

[9] VMware Virtualization[EB/OL]. http://www.vmware.com/

[10] 刘鹏. 云计算[M]. 北京:电子工业出版社, 2010

[11] System Center Virtual Machine Manager [EB/OL]. http://www.microsoft.com/zh-cn/server-cloud/system-center/virtual-machine-manager.aspx

[12] VMware vCenter Multi-Hypervisor Manager[EB/OL]. http://www.vmware.com/support/mhm/doc/vcenter-multi-hypervisor-manager-10-release-notes.html

[13] HotLink SuperVISOR[EB/OL]. http://www.hotlink.com/

[14] openstack[EB/OL]. http://www.openstack.org/

[15] AWS SDK for Java Developer Guide [K]. Amazon Web Services, Inc. 2012

[16] Jin S. VMware VI and vSphere SDK: Managing the VMware Infrastructure and vSphere 1st[M]. Prentice Hall PTR Upper Saddle River, NJ, USA, 2009

[17] Performance Counters[Z]. VMware, Inc. Nov, 2008

[18] Amazon Cloud Watch Developer Guide [K]. Amazon Web Services, Inc. 2013