利用元数据建模构建可扩展的灵活系统*)

欧金梁　金茂忠
(北京航空航天大学软件工程研究所　北京100083)

摘　要　软件需求的复杂化对现代软件的构建技术提出了很高的要求。构件技术在一定程度上解决了这个问题,但它很难处理不断扩充的功能与原有系统的适配与连接。本文提出了元数据建模的方法,通过使用基于XML描述的元数据并且利用某些面向对象语言本身的自省机制(如Java的反射机制)来连接构件和基本框架的技术,以达到构建灵活系统的目的。本文同时给出了北航软件工程研究所使用元数据成功构建可扩展的QESAT(软件分析与测试工具)的示例。

关键词　元数据,连接器,构件,框架,可扩展性

A Method of Metadata Modeling to Construct Extensible and Flexible Systems

OU Jin-Liang　JIN Mao-Zhong
(Software Engineering Institute, Beihang University, Beijing 100083)

Abstract　Due to ever increasing complexity of the software requirement, it is now becoming harder to construct modern software. Although present component technique is useful in some ways in solving this problem, it leaves pendent the connection and adaptation between the constantly expanded functions and the origin system. Proposed in this paper is a method of metadata modeling to construct flexible systems, in which the XML-based metadata and the reflection mechanism of some object-oriented languages, e.g. Java, are applied to connect both the expended components and the core framework. A successful demonstration of metadata modeling for an extensible software QESAT(Quality Engineering : Software Analysis and Test Tool) is also presented.

Keywords　Metadata, Connector, Component, Framework, Extensible

随着计算机应用的日趋复杂,对软件的要求越来越高。软件需要不断地适应复杂、多变的需求。软件工程从许多方面,例如OOP、OOA/OOD、各种软件过程(RUP,XP)和方法等对此进行了研究。这些技术运用得当,可以构建出可用、可靠、稳定的系统。但是,很多技术的应用场景是从头开始构建一个符合用户需求的软件。也就是构建一个单个的系统。然而在商业环境中,软件经常成为一个系列,也就是一个软件产品簇[1]。这些软件的特点就是它们具有很大部分的相似处,其体系结构和核心部件基本是相同的,不同的只是一些新增的功能点和特性。还有一些软件需要开放的结构,以利于对软件的功能进行扩充,包括第三方的扩充。对于这些软件来说,仅仅利用以上的技术是不能很好地适应这些要求的。和传统的软件开发方式不同,这些软件的开发需要对所属的行业和领域进行领域的分析,然后构建一个核心系统,在核心系统的基础上进行扩展,就构成了所需的不同的软件产品。领域工程、特征建模以及框架技术都是为了构造软件产品簇所提出的方法和技术[2]。

本文的其他部分说明了软件的抽象分解和合成能力、抽象和合成技术,提出了一种元数据建模的方法,并论述了元数据是连接软件不同部分以构造灵活系统的较好的方案。最后文章还给出了北航软件工程研究所利用元数据构建可扩展系统的实例。

1　软件的抽象和部件的合成

构建软件簇或一个可扩展的系统要求有一个可复用的核心部件和一些实现特定功能的扩展部件。这需要对系统进行良好的分解和抽象。良好的抽象边界和粒度有利于简化对软件系统的建模,并且易于进行软件合成。面向框架的开发技术中提出了通过领域分析将软件系统中的固定部分和可变部分进行分离[3]。特征建模提取软件的共同特性和可选特性[2]。

合成是将不同的模块组合以构建完整的软件系统。可合成性和合成能力是可复用部件的重要性质。合成代价不仅与抽象边界相关,要求模块有较高的内聚性和模块间有较低的耦合性;还和合成技术密切相关,不好的合成技术在模块间添加厚的粘合层来配接模块间的接口,不仅使系统变得复杂,还可能导致潜在的不稳定性。

1.1　软件簇的抽象技术:框架和构件

面向对象技术是创建可复用组件的强有力的技术。而面向对象的框架和构件技术为构建软件产品簇提供了很好的条件。

和传统的单一产品的软件开发不同,软件产品簇的开发通常是基于可复用的框架的,通过框架和不同的构件的组合,构成了不同的软件产品。

为了将框架和功能部件(构件)连接在一起构成一个完整的系统,框架和具体的功能部件之间必须有清晰的接口。根据文[3],一个框架由稳定部分(或叫frozen spot)和可变部分(或叫hot spot)组成。可变部分通常就是对外提供的接口。不同的产品有不同的功能部件,这些功能部件通常都要符合框架所提供的接口。这些接口和连接框架与功能部件的方式是构建可扩展的灵活系统和软件产品簇的重要手段。

1.2　连接和扩展方式

好的合成方式要求部件之间的连接点是简单和清晰的。

*)项目资助:国家"863"高技术计划资助项目:"软件测试技术与软件测试平台",课题编号为:2001AA113100。欧金梁　硕士研究生,研究方向为软件工程和软件测试。金茂忠　教授,博士生导师,研究方向为编译、软件工程、过程工程、并行处理和软件测试。

而对于开发可扩展的软件产品簇来说，这些连接点就表现在核心对外提供的接口(interface)，也可以称作扩展点。简单和清晰同样是对扩展点的要求。如何有效地表达它们呢?在目前应用广泛的开发方式中，利用面向对象语言的多态特性进行扩展和基于规则的解释执行是两种常见的方式。

1.2.1 基于面向对象语言的多态特性进行扩展 在面向对象的语言中，多态和动态捆绑使得类型可以延迟到运行时进行绑定。对象的实际行为和运行时它的实际类型相对应，这为抽象设计和扩展提供了更多灵活性[2]。多态性使得开发稳定的没有差异的框架代码成为可能，而具体的运行时行为则由扩展的功能部件改写框架提供的接口实现。

1.2.2 基于规则的解释执行 解释型语言具有很好的动态特性。规则是对事实的描述，解释器通过推导原则对规则进行解释执行。不同的规则组合构成了不同的系统。这种方式依赖于一个强大的解释引擎和对规则的良好的定义。在这种方式中，可复用的核心部件就是解释器，功能的实现完全是通过不同的规则来表达的。

这两种方式都有它们的优缺点。对于第一种，一致的核心工作流程以及只需要扩展具有良好定义的接口使框架的设计要容易些，也比较稳定。但要建立一个完整可运行的系统，通常需要通过编写代码来实现，一个粘合层将扩展的功能部件实例化并将其连接到核心部件上。粘合层的质量对整个系统的正确运行有较大的影响。对于第二种，表达直观，灵活性强是它的优点。但是这种系统需要对规则的定义和解释器的能力都有较高的要求。从一个极端上看，解释型程序设计语言、函数式语言或逻辑语言，都可以构造相当灵活的系统，但是开发的工作量是很大的，并且没有充分利用与领域相关的具有很高复用价值的核心框架。如果规则和具体领域密切相关，则对解释器的要求就相当高。另外，解释执行的方式具有执行效率低的缺陷。

2 元数据建模和基于 XML 的元数据表示

为了利用上述两种方式的优点并避免它们的缺点，我们提出了将两者进行结合的方法。首先构造可复用的面向对象框架，利用多态特性声明可扩展的接口。然后在框架与扩展功能部件的连接、子类型的实例化过程中利用解释引擎(也称作连接器)在运行时动态完成。

要将框架和扩展功能部件进行良好的合成，需要对连接点进行建模。通过对连接点仔细的建模，有可能对具体的问题域构造一个通用的解释引擎。通常连接点都清晰地表明了该系统要完成的功能的概要的描述，也就是一些文献[3]中说的框架中的热点(hot spot)。在本文中，我们使用元数据对连接点建模。

为了进行动态的连接和类型的实例化，需要编程语言提供支持，主要是反射特性。以下分别进行说明。

2.1 元数据建模

当数据在程序中不是被加工的对象，而是被用来对程序的运行起控制作用，并且可以通过值的改变而改变程序的行为时，这样的数据称为元数据(metadata)[5]。元数据本身是一种数据，可以对其进行操作和修改，这样，使得程序在稳定的未经修改的情况下，通过修改元数据的方式，可以改变程序的动态行为。

元数据的应用广泛。国内外对元数据的关注主要集中在：把它作为实现数据字典的方式、数据交换的标准、资源管理(广泛应用在图书馆、地理信息系统等)；或者用它描述程序的执行系列[6]。

在系统中利用元数据来处理人机的互操作，功能模块配置等[7,8]不是新的想法。但这通常只对单一系统的部分特性进行描述，以便在运行时进行选择。本文将这一思想扩展到构建软件簇的领域中，利用元数据来描述框架和扩展部件的连接点，以获得一个松耦合的可扩展系统。

利用元数据来描述连接点，并给定具体的连接点的描述规范就是元数据建模。扩展部件根据规范(元数据建模的结果)提供元数据，在程序运行时，由嵌入在框架中的连接器解释元数据，利用语言提供的动态加载和连接机制，将扩展部件连接到框架中，构成了一个完整的系统。基于元数据建模、可复用框架和扩展部件来构建灵活可扩展的系统或产品簇的过程如图1所示。

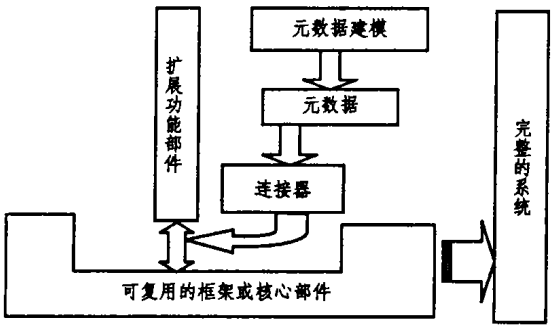

图1 利用元数据建模构建系统的过程图

2.2 基于 XML 的元数据

XML(Extensible Markup Language)是由 W3C 于1998年2月发布的一种标准，是一种元标记语言。它的标记可以由使用者根据需要扩展，其结构是一种严格定义的树型结构。它以一种开放的自我描述方式定义了数据结构，在描述数据内容的同时能突出对结构的描述，从而体现出数据之间的关系。

它的优点主要有，可扩展性：它允许各个组织、个人建立适合自己需要的标记集合；灵活性：数据存储格式不受显示格式的制约。

另外，随着 XML 的应用的广泛，XML 的解析器已经非常成熟。因此利用 XML 来表示元数据有着很大的优势。首先，XML 的描述能力很强，其中的可扩展的标记为描述各种类型的元数据提供极大的方便。其次，成熟的 XML 解析器使得构造连接框架和扩展部件的连接器变得更为简单。

2.3 程序设计语言的支持(反射机制)

要完成框架和扩展部件的动态创建和连接，离不开语言的支持。因为扩展的对象和操作都是以数据的方式存放在元数据中，只是在运行的时候根据需要创建和执行它们，因此要求程序设计语言能够在运行时而不是编译时根据类型名动态创建对象，查询该对象所能执行的操作并执行之。Java 就可以满足我们的要求，在 Java 中，这称作反射(reflection)机制。

2.4 建模过程

基于前述内容，我们给出的元数据建模过程如下：

a)对系统进行分析，抽象出系统的稳定部分和可扩展部分，构建稳定的框架。

b)对可扩展部分采用抽象类或接口类的方式声明扩展点，以供扩展。

c)对扩展点进行元数据建模。

d)构造连接器并嵌入到框架中。

e)扩展部件实现扩展点,并给出以 XML 描述的元数据。

f)框架在运行时通过连接器解释元数据,连接、加载和运行扩展部件。

3 实例研究

QESAT (软件分析与测试工具)是北京航空航天大学软件工程研究所承接的863课题下的子课题。以下讨论元数据在该系统中的应用。

3.1 元数据作为连接方式实现表示和逻辑的分离

在现代的 GUI 程序中,程序和用户的交互比较频繁。用户可以通过菜单、按钮、工具条、热键等来和程序进行交互。在很多应用程序的开发中,尤其是利用可视化开发工具进行快速原型的开发时,创建菜单、按钮等的工作和它们的对应需要执行的动作都是硬编码在程序当中的。这种方式,对于小型的单一的系统自有它的优势。但是对于大型程序和软件产品簇的开发,这种方式有着巨大的缺点。首先是表示和逻辑的不分。其次,不同的界面形式(菜单或按钮等)可能对应于同一个动作,采用这种方式可能造成潜在的代码冗余。第三,为了统一这些相同的动作,需要将这些和界面相关的元素统一到一个大型的类中,假设取名叫 MainFrame,则在 MainFrame 中集中了大量不同用途的方法 。第四,造成了完成具体动作的类和布置界面元素的类之间有很强的依赖关系。第五,当功能进行扩充时,牵涉到很多的代码变动。

实际上,利用 Command 模式[4]可以将这些界面元素对应的操作封装到不同的类中。在 Java 中,可以将它们抽象为 Action 类。每一种具体的动作都对应一个具体的 Action 的实现。在每个菜单或按钮中可以关联一个 Action 的实例。这样,基本实现了界面布局和具体功能的解耦。

更进一步,可以将布局也从 MainFrame 中分离出来。通过元数据建模,可以将 MainFrame 做成一个通用的界面布局引擎。具体做法是:一个 Action 的子类完成具体的动作,元数据描述该动作在界面的表现(如菜单项和工具条按钮的名称,图标,热键等)和位置,MainFrame 解释这些数据并将它们在界面上表示出来。利用这种方式,将不同的部分很好地分离,MainFrame 也变得相当稳定,功能部件的扩充时只需要实现不同的 Action 类并在元数据中进行说明,而不影响到框架代码。图2是元数据中描述一系列 Action 的片断:

```
〈ActionSetDefine id="seibuaa. qesat. project"
  menu="Project"〉
  〈ActionItem
    class="seibuaa. qesat. project. action. NewProjectAction"
    text="New Project"
    tooltips="Create a New Project"
    icon="/icon/newprj. gif"
    mnemonic="N"
    accelerator="ctrl N"
  /〉
  〈ActionItem
    class="seibuaa. qesat. project. action. OpenProjectAction"
    text="Open Project"
    tooltips="Open an exist Project"
    icon="/icon/openprj. gif"
    mnemonic="O"
    accelerator="ctrl O"
  /〉
  〈!-- … --〉
〈/ActionSetDefineHT〉
```

图2 描述 Action 的元数据片断

图2中,ActionSetDefine 描述了一个 Action 集合,通常包含紧密相关的几个 Action。它有属性 id(全局唯一的标识)和 menu(表示加载时位于哪个菜单下)。在本例中的 ActionSetDefine 包含三个 Action,每一个 Action 用一个 ActionItem 的元素表示。在 ActionItem 的属性中,class 表示具体执行动作的类,该类的实例将在运行时动态创建。其他属性描述该 Action 的显示数据。

3.2 基于 XML 的元数据描述软件测试过程的执行

软件的测试通常由一系列的步骤组成。不同的测试通常都有不同的过程。在本系统中集成了 C++和 Java 的项目,每种项目都有不同的测试类型。如普通的覆盖率测试、内存检测分析、数据流分析等。这些过程之间的差别比较大,因此在程序中对这些过程的执行顺序进行编码将导致过程的固化,任何一种新的过程的添加都将导致编写新的过程类和将其集成和管理的工作。过程的创建和管理相当复杂。实际上,我们注意到,过程的每个步骤其实是一个个的任务,这些任务是可以重用的,可以通过实现一些任务类来完成具体的工作。而过程管理的复杂性主要体现在各个步骤的顺序性和它们之间的依赖关系。这恰恰就是元数据擅长的领域。我们可以利用元数据对过程的任务的执行顺序和依赖关系进行建模,利用执行引擎解析它们之间的依赖关系并创建具体的任务来执行它们。考虑到在开放源代码领域,Ant[9]作为一种应用广泛的构建工具,已经符合我们的管理测试过程的需求,因此在我们的项目中,采用 Ant 的 build 文件的格式作为过程元数据的规范,Ant 作为内置的执行引擎(需要经过包装以适应我们的项目)来处理测试过程的执行。

经过实践,本方案取得很好的效果。对新的测试过程,只需要定义新的过程模板。在程序中创建新的测试时,根据用户的配置和给定的过程模板,生成符合 Ant 规范的 build 文件,内置的 Ant 引擎就可以根据需要执行具体的过程了。

3.3 用 XML 描述状态转换图自动构建有限状态自动机

在进行数据流分析时,经常会遇到语句或变量的系列问题。对于给定的系列,需要判断它们是否符合(或不符合)某个规则。为了简化,可以将系列符号化,一个特定的语句(或变量)在一个特定的分析场景下用某个符号表示。一个给定的语句系列就是一个特定符号串。可以构造一个和规则对应的自动机来识别给定的符号串是否合法。

假设我们有状态集合 $Q\{INIT, DEC, REF, ASS, EXC\}$,有输入符号集合 $\sum\{a, r, u\}$和转换表 δ 如下。

表1 状态转换表

	a	r	u
INIT	ASS	REF	DEC
DEC	ASS	EXC	EXC
REF	ASS	REF	DEC
ASS	EXC	REF	EXC
EXC			

那么我们可以构造一个自动机 $A=(Q, \sum, \delta, INIT, \{DEC, REF, ASS\})$,来识别非法系列。在我们的系统中,对于符号串中的每一个符号,在自动机的当前状态中都有转移到下一个状态的路径,并且在所有输入符号都完成后,状态在终止状态集合$\{DEC, REF, ASS\}$中,那么该符号串是合法的,否则是非法的。根据这个自动机,可以判别符号串"aruau"是非法的,导致其最后进入 EXC 状态中,这不是合法的终止状态。而"rarua"是合法的。

对于这种问题的解决办法之一是:编码每一种状态和与该状态下的给定的合法输入符号对应的下一个状态。这实际是手工编码构造一个特定的自动机。对于小的规则,或者系统中只存在一种规则,这是一种可行的方法。它比较直观,简明。但是系统中规则显然不只一种,并且有些规则是复杂的。那么,自动构建自动机就显得相当必要了。

根据元数据建模的步骤,本问题的可变部分就在于自动机本身。因此需要对自动机进行建模。一个自动机包含一个状态集合,一个输入符号集合,一个转移集合和一个初始状态。图3给出了一个简化的自动机的元数据片断。

```
〈AutoMata〉
 〈StateDefine〉
  〈State name="INIT" description="start auto machine"/〉
  〈State name="DEC" description="variable declaration"/〉
  〈State name="REF" description="reference"/〉
  〈State name="ASS" description="assignment"/〉
  〈State name="EXC" description=" something error"/〉
 〈/StateDefine〉
 〈TransitionDiagram startState="INIT"〉
  〈StateItem state="INIT"〉
  〈Transition input="a" nextState="ASS"/〉
  〈Transition input="r" nextState="REF"/〉
  〈Transition input="u" nextState="DEC"/〉
 〈/StateItem〉
 〈StateItem state="DEC"〉
  〈Transition input="a" nextState="ASS"/〉
  〈Transition input="r" nextState="EXC"/〉
  〈Transition input="u" nextState="EXC"/〉
  〈Transition input="X" nextState="INIT"/〉
 〈/StateItem〉
   〈!--……--〉
 〈/TransitionDiagram〉
〈/AutoMata〉
```

图3 一个简化的自动机的元数据片断

利用元数据,我们实现了自动机构建的自动化。定义新的规则,只需要一份描述该规则的元数据,利用自动机引擎,就可以动态建立一个新的自动机来识别新的规则。这样就实现了灵活的扩展。

总结和展望 本文通过对构建软件产品簇的过程进行分析,提出框架和构件是建立可复用的灵活系统的基本部件,并且提出了基于 XML 描述的元数据作为粘合剂连接框架和构件的方式。实践证明,这种方式构造的系统具有足够的可扩展性和实现的简单性。在北航软件工程研究所开发的基于框架和插件技术的可复用软件测试平台中,元数据建模和以元数据连接不同部件的技术是核心和基础技术之一。我们相信,在该平台中,元数据技术将得到很好的应用和发展。

参 考 文 献

1 Czarnecki K, Ulrich W. Eisenecker Generative Programming : Methods, Tools, and Applications Addison Wesley/Pearson, 2000

2 Mili H, et al. Reuse-Based Software Engineering Techniques, Organization, and Controls by John Wiley & Sons, 2002

3 Parsons D, Rashid A, Speck A. A "framework" for object oriented frameworks design; Telea, A. ; Technology of Object-Oriented Languages and Systems, 1999. In: Proc. of, 1999. 141～151

4 Gamma E, et al. Design Patterns: Elements of Reusable Object-Oriented software Addison Wesley/Pearson, 1994

5 Bretherton F P, Singley P T. Metadata: a user's view Scientific and Statistical Database Management, 1994. In: Proc. Seventh Intl. Working Conf. on, Sept. 1994. 28～30

6 Atarashi R S, Kishigami J, Sugimoto S. Metadata and new challenges Applications and the Internet Workshops, 2003. In: Proc. 2003 Symposium on, Jan. 2003. 395～398

7 Pittas N, Jones A C, Gray W A. Evolution support in large-scale interoperable systems: a metadata driven approach Database Conference, 2001. ADC 2001. In: Proc. 12th Australasian, 2001. 161～168

8 de Carvalho Moura A M, Estevao da Silva L A. Machado Campos M L. A metadata approach for designing configurable interfaces in digital libraries Database and Expert Systems Applications, 2001. In: Proc. 12th Intl. Workshop on, Sept. 2001. 942～947

9 http://ant.apache.org

(上接第192页)

而有$\langle c, i_p \cup o_p \cup s_p \rangle \notin Y$。

综合情况一、情况二,(c)得证。

总结及今后的工作 AO-RT-Z 扩展形式化规格说明 RT-Z 于面向方面的实时组件设计,提供了对 AOP 的有效支持,利用形式化方法所固有的推理证明机制和辅助工具,能够保证实时和嵌入式系统的重要组件规格和设计的无二义性,方面联结的支持保证了系统模块分析的独立性、简洁性和正确性,为 AOP 提供了有力的支持。

专门的实时语言通常都能具有对实时和嵌入式系统各方面的独立编程支持,因此将 AO-RT-Z 同这些语言相结合构建一个完整的开发框架具有良好的应用价值。

UML、AOP 技术及形式化的结合,充分利用 AOP 的思想和 UML 的强大的建模能力和形式化方法的严谨性及语义,是今后研究的一个方向。

把时间分离出来构造为一个时间方面,建立一个比较完整的时间模型来建模系统时间,促使时间方面与系统其它方面分开,能够进行单独的时间方面设计,同时又能够根据需要把设计好的时间方面织入到系统中,从而简化了实时系统建模的复杂性,是今后的另一个研究方向。

参 考 文 献

1 The AspectJ Programming Guide. Xerox Corporation ,September 2002. Available at: http://aspectj. org/doc/dist/progguide/index. html

2 任洪敏,钱乐秋. 构件组装及其形式化推导研究. 软件学报,2003, 14(6):1669～1677

3 Suhl C. RT-Z: An integration of Z and timed CSP[R]. In: [AGT99], 1999. 51～65

4 Suhl C. Applying RT-Z to develop safety-critical systems[C]. In: Proc. of the Third Intl. Conf. on Fundamental Approaches to Software Engineering (FASE 2000), number 1783 in Lecture Notes in Computer Science, Springer-Verlag, 2000. 51～65

5 ZUM'98. The Z Formal Specification Language[C]. number 1493 in Lecture Notes in Computer Science, Springer-Verlag, 1998. 5～23

6 Andrews J H. Process-algebraic foundations of aspect-oriented programming. In: Proc. of the Third Intl. Conf. on Metalevel Architectures and Separtion of Crosscutting Concerns, volume 2192 of Lecture Notes in Computer Science, Benlin: Springer-Verlag, 2001. 187～209

7 RAISE Language Group. The RAISE Specification Language[C]. BCS Practitioner Series. Prentice-Hall, 1992

8 陈广明,陈生庆,张立臣. Z 实时扩展及基于多视点的应用模式. 计算机应用,2005,25(2)

9 ftp://ftp. irt. uni-hannover. de/pub/pearl/report. pdf(in English)

10 Fischer C. How to combine Z with a process algebra. [R]. In J. P. Bowen, A. Fett, M. G. Hinchey, editors ,2000

11 Stankavic J, Zhu R, Poornalingam R, et al. "VEST": an aspect-based composition tool for real-time systems. In: Proc. of the 9th Real-Time Applications Symposium 2003, Toronto, Canada: IEEE Computer Society Press, May 2003. 110～123

12 stoyenko A, Marlowe T. Polynomial-Time Transformations and Schedulablity Analysis of Parallel Real-Time Programs with restricted Resource Contention Real-Time Systems. Fall 1992. 307～329

13 Schneider S. An Operational Semantics for Timed CSP[C]. number 1453 in Lecture Notes in Computer Science, Springer-Verlag, 1997. 45～61