扩展方面机制的软件产品线体系结构建模及构件组装实现

沈立炜 ， 彭 鑫 ， 赵文耘

（复旦大学计算机科学技术学院软件工程实验室 上海 200433）

摘 要： 软件产品线是提高软件开发效率与质量的有效途径，它以体系结构(SA)为蓝图，定义组成产品线的构件与构件之间相互作用的关系，指导基于构件的应用产品组装实现。现有的基于接口连接式的体系结构仅能描述构件间的直接交互，却无法支持产品线可变性所带来的更为复杂的构件交互情况。因此，本文提出一种扩展方面机制的软件产品线体系结构建模及构件组装实现方法，其核心是一套扩展 xADL2.0、结合面向方面机制的软件产品线体系结构描述语言。它能支持基于可变性的产品线体系结构设计与定制，并指导应用产品的构件组装过程。在此方法的基础上，我们开发了原型工具 FdSPLC，提供对体系结构的可视化建模以及应用产品的自动化生成。

关键词： 软件产品线开发；软件体系结构；构件组装；构件交互模式

中图分类号： TP311 **文献标识码：** A **文章编号：** 0372-2112

Software Product Line Architecture Modeling and Component Composition Implementation with Extension of Aspectual Mechanism

SHEN Li-wei , PENG Xin , ZHAO Wen-yun

(School of Computer Science, Fudan University, Shanghai 200433, China)

Abstract: Software product line (SPL) can increase the efficiency and quality of software development. Software architecture (SA), as the blueprint of SPL, defines the inter-relationships between components and guides the component composition implementation. However, the existing interface connection architecture is limited to describe the direct interactions between components. It cannot support the more complex interaction situations which emerge with the SPL variability. In this paper, we propose a method of software product line architecture modeling and component composition implementation with extension of aspectual mechanism. The core is an architecture description language (ADL) which extends xADL2.0 and combines with aspect-oriented techniques. The ADL supports the design and customization for SPL architecture based on variability, and instructs the component composition process for applications. Furthermore, we have developed a prototype tool FdSPLC which provides the visual modeling of architecture as well as the automatic application derivation.

Key words: Software product line development; Software architecture; Component composition; Component interaction style

1 概述

在以构件为基本单元的软件产品线中，体系结构作为整个开发过程的蓝图，定义了组成产品线的构件与构件之间的相互作用关系[1]，它包括领域体系结构(DSSA)与应用体系结构(ASSA)：DSSA 是领域工程的制品，描述了所有应用系统的共性与差异性，而 ASSA 则是在应用工程阶段由前者定制、裁剪得来。

软件体系结构为构件的集成组装提供了基础和上下文[2]，文献[3]比较了三种不同类型的体系结构，包括对象连接式、接口连接式与插头插座式。其中，接口连接式体系结构通过接口定义系统中构件之间的所有连接，匹配所要求的功能和所提供的功能，因此降低了构件之间的依赖性，提高了构件的独立性和可复用性[4]。这种灵活的连接方式有利于软件产品线的设计与组装，尤其对于具有实现变体的构件而言，在体系结构设计时即可通过接口建立构件之间的关联关系，然后在组装实现时确定符合接口描述的构件实现体。

软件产品线的 DSSA 含有可变性，与单系统的体系结构相比较，它包含更为多样的构件交互情况。其中，某些构件交互无法用接口连接式的体系结构

*)基金项目：国家863计划(No.2007AA01Z125；No.2006AA01Z189)；国家自然科学基金(No.60703092).

描述，因此，我们仍然需要系统化的方法支持软件产品线体系结构的建模，同时指导自动化的应用产品生成。

针对以上需求，本文提出了一种扩展方面机制的软件产品线体系结构建模及构件组装实现方法，其核心是一套扩展 xADL2.0[7]、结合面向方面机制的软件产品线体系结构描述语言。它包括对不同构件交互类型的定义，支持基于可变性的产品线体系结构设计与定制，并指导应用产品的构件组装过程。在此方法的基础上，我们开发了原型工具 FdSPLC。它提供对体系结构的可视化建模，另外支持应用产品的自动化生成。

本文剩余结构如下：第2节是背景介绍，分析产品线实现所存在的问题。第3节介绍扩展xADL2.0的产品线体系结构建模技术，包括对不同构件交互模式的定义。在第4节中基于产品线体系结构的应用产品组装将被介绍，包括构件交互模式的实现过程。第5节是原型工具FdSPLC的简要介绍。最后是总结以及对未来工作的展望。

2 背景介绍

2.1产品线体系结构

可变性是软件产品线研究的核心，它将产品线的 DSSA 划分为基础部分与可变部分。基础部分提供的功能(基础程序)是被所有应用产品所共享的；而可变部分的功能代表了应用产品之间的差异性。一般而言，可变部分是对基础部分功能的扩展，根据应用需求决定是否附加到基础程序上去。DSSA 中构件的可变性分为 Optional、Alternative 与 Abstract 三种类型：Optional 表示可选构件，即应用系统体系结构中可以包括该构件也可以移除该构件；Alternative 表示多选一构件，即 DSSA 中为某一构件提供了多个实现变体，应用系统体系结构可以从中选择一个；Abstract 表示抽象构件，即以占位符的形式存在于 DSSA 中且没有提供任何实现变体的构件，抽象构件的具体实现将在应用系统开发时提供。

体系结构描述语言(ADL)负责软件体系结构的描述、分析与重用[5]。对于软件产品线，ADL 必须明确描述 DSSA 中构件的可变性。国内外已有很多工作关注于产品线的 ADL，其中 xADL 2.0[7]是一种高度可扩展的、基于 XML 的体系结构描述语言，通过一组 Schema 支持产品线体系结构的建模。Structure&Type Schema 是 xADL2.0 中最重要的部分，它用来描述产品线设计时(design-time)的基本元素，包括构件(component)、连接器(connector)以及关联(link)。体系结构中的每一个构件能被指派一个构件类型(componentType)，构件类型定义了构件的类型信息，例如型构(signature)等。Options 和 Variants Schema 用来定义产品线体系结构的可变性。其中 Options Schema 表示 Optional 可变性，而 Variants Schema 能够支持 Alternative 与 Abstract 可变性的描述(如果 variant 构件至少包含一个变体，它具有 Alternative 可变性；如果不包含任何变体，则其具有 Abstract 可变性)。每个 Optional 或者 Alternative 变体元素都伴随一个 Guard 条件。当条件被满足时，Optional 元素或 Alternative 的变体元素将被包含在体系结构中。另外，Alternative 构件的变体之间具有互斥的 Guard 条件，表示在同一时刻至多有一个变体被选择。xADL2.0 的详细描述可参见文献[7]。

2.2产品线实现问题分析

在本小节中，我们以简化的图书馆管理系统产品线实例来分析现有的基于接口连接式体系结构的不足。图 1 是该产品线的初始体系结构图。其中，LibraryMainForm 是系统的主界面，通过它可以进入借书界面(BorrowBookForm)与还书界面(ReturnBookForm)。借书、还书的过程都将与图书信息管理(BookManage)和读者借阅记录管理(ReaderRecordManage)的功能交互。还书的流程还包括罚金计算与收取，其支付模式具有 Alternative 可变性，变体分别为现金支付(PenaltyByCash)与学生卡支付(PenaltyByStudentCard)。另外，记录日志(Log)、获取图片(getImage)与图书图片显示(BookPicShow，表示借书界面是否显示图书封面图片，它将使用获取图片功能)是 Optional 的功能，它们依照应用需求被包含进或排除出实际系统。

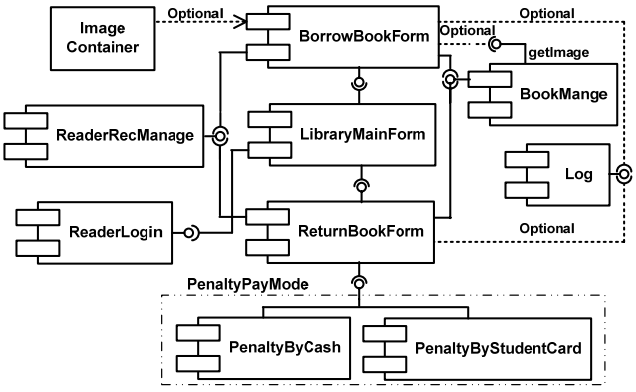

图 1 图书馆管理系统产品线初始体系结构

当采用接口连接式体系结构建模时，将出现以下不适合描述或无法描述的情况：

(1)不适合描述更改执行流程的构件交互。Optional可变性会影响系统的执行流程(如果Log被选定，借书或还书的执行过程将会包含对Log的调用，反之则不包含)。从实现的角度看，接口连接式的体系结构要求与Optional功能连接的构件必须包含与其交互的接口，同时对接口的控制必须在构件内部实现(Optional功能选定时使用此接口，否则取消使用)。一般而言，可以通过条件判断语句(if/else)进行控制，但它会对构件实现提出很高的要求，也不利于构件的复用。

(2)不适合描述需求的分散特性。产品线的需求被分解

为责任(responsibility)[6]，这些责任分散在不同的实现中(图书图片显示功能被分解为图片框ImageContainer与BookManage构件的getImage功能)。接口连接式体系结构中的构件一般表示单个需求的功能，因此它不适合表达具有分散特性的需求。

(3)无法描述体系结构中Optional资源。图片框ImageContainer作为完成Optional功能“图书图片显示”的资源角色(resource role)[9]，需要被加入借书界面中。在这种情况下，资源也具有Optional可变性。我们把资源看作构件，但它与目标构件之间的交互关系无法用接口连接的方式描述。

2.3 AOP在产品线开发中的应用

AOP 技术大大提高了软件开发的模块化(尤其是针对横切方面)程度[8]，而这一改进主要源于 AOP 中方面的两大特性：多量化(Quantification)和不知觉性(Obliviousness)。目前对于 AOP 技术的应用一般都强调它对横切关注点的支持能力(通过多量化特性)，但 AOP 的“不知觉”却为构件的交互提供了有效的帮助。对于更改执行流程的构件交互，采用 AOP 在不影响基础程序的情况下，将 Optional 的功能编织到基础程序上是一种理想的解决方案；另外，对于 Optional 的资源交互，AOP 的 Introduction 机制能够将资源插装到目标构件中，而不需更改目标构件的实现。在我们的前期工作[9]中，我们通过 AOP 实现应用产品生成。在本文的方法中，我们将以接口连接式的体系结构为主体，结合面向方面的机制支持产品线的描述与开发。

3 扩展 xADL2.0 的产品线体系结构建模

3.1 产品线体系结构中的构件交互模式

构件交互模式表达了构件之间的相互作用关系，同时指明了通过何种方式将构件的功能合并。我们的方法定义了三类交互模式，分别为**接口连接模式**、**编织模式**与**引入模式**。

接口连接模式的思想来源于接口连接式体系结构[3]。它通过定义接口间的直接关联，描述构件之间稳定的交互。一般而言，DSSA中基础部分的构件交互清晰明确，不会改变，因此适合用这种模式表示。另外，DSSA的Alternative与Abstract可变性体现在构件的具体实现上，不影响执行流程，这些可变构件对外的交互关系是固定的，因此与基础部分的构件交互类似，也适合采用接口连接模式描述。

编织模式采用 AOP 的思想，描述引起执行流程变化的变体与其它构件之间的交互。实现 Optional 功能的构件属于这一类。此时 Optional 可变性体现在此构件接口与其它接口的交互关系上(例如，BookManage 含有多个接口，某些接口提供的功能属于基础程序，因此 Optional 不应代表构件的性质)。

表1 扩展xADL2.0的体系结构描述语言元素介绍

FDSPL	描述产品线DSSA设计时(design-time)的基础部分。
Component	构件是Structure层上元素，是组成产品线的基本功能单元。包含一组服务接口(Provide Interface)与请求接口(Require Interface)，每一个接口由一组方法声明(Method)所构成。
Connector	连接器表示**接口连接模式**的构件交互。 连接器负责构件之间的逻辑通信，由一对服务角色(Service Role)与请求角色(Request Role)构成，服务角色指向构件请求接口，请求角色指向构件服务接口。方法映射(Method Mapping)定义了接口之间需要匹配的方法与方法参数。
ComponentType	构件类型是Type层上元素，反映了构件的型构(与构件接口对应)等信息。同时构件类型能够关联到具体的实现体，其类型(如java类、ejb构件)可通过进一步扩展Schema来定义。另外，一个构件类型能够指派多个构件，表示Structure层上的不同构件能够具有相同的实现体。
FDOptions	描述产品线DSSA的Optional可变性及其与构件的交互。
Weaver	编织器表示**编织模式**的构件交互。 编织器将构件接口功能编织到连接器所包含的接口交互上。编织器定义编织目标点，包括在交互之前(before)，在交互之后(after)以及在交互的外围(around)；另外，编织器定义上下文映射(Context Mapping)，使得编织进的方法能够从已有交互中得到参数，保证正常运行。 编织器具有Guard条件，表示其Optional可变性。
FDVariants	描述产品线DSSA的Alternative或Abstract可变性。
Variant－ComponentType	构件类型VariantComponentType继承自ComponentType。当它至少包含一个变体(Variant)时，此构件类型指派的构件具有Alternative可变性，当它不包含任何变体时，构件具有Abstract可变性。
Variant	变体关联到构件类型，表示变体的具体实现。同时，变体间拥有互斥的Guard条件，当某个变体的条件满足时，只有该变体的实现将会被纳入应用产品。
FDIntroduce	描述产品线DSSA中的Optional资源及其与构件的交互。
Resource	资源的实现体、实例化方法以及其他附加方法。
Introduce	Introduce关联表示**引入模式**的构件交互。 包含资源在目标构件中的插装点(pointcut)定义，以及表示可变性的Guard条件。

图2是编织模式的示意图，表示一个提供Optional功能的构件接口通过编织器(Weaver)被编织到已有的接口交互中。Optional特性体现在编织器上，一旦Optional功能未被选定，编织器将会被移除，这种插装式的交互也不复存在。在图书馆管理系统的实例中(图1)，提供Optional功能Log的接口即可通过编织器被插装到借书/还书界面与读者借阅记录管理的交互上。

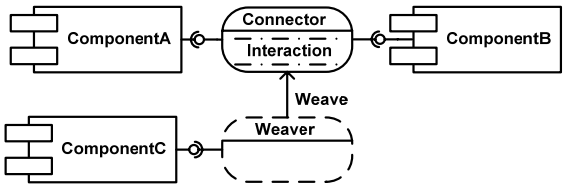

图 2 编织模式的示意图

引入模式主要解决Optional资源的交互问题。此模式在不影响目标构件实现的前提下，采用AOP的Introduction机制将Optional资源本身及其实例化代码插装到构件中。图1实例中的ImageContaier与BorrowBookForm之间的交互属于引入模式，在实现阶段此资源将被封装为AOP文件。

3.2 扩展xADL2.0的产品线体系结构描述语言

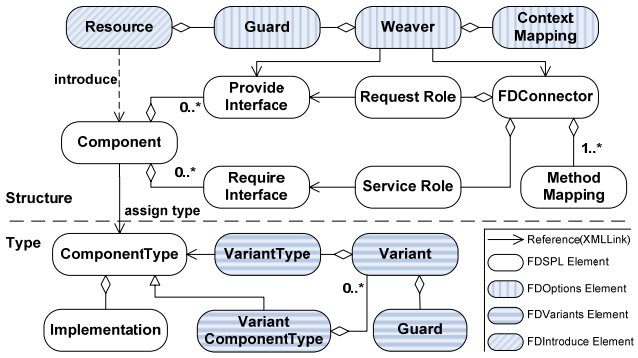

图 3 扩展 xADL2.0 的 ADL 的元模型

本方法中的 ADL 继承 xADL2.0，重定义其中的部分元素，并且扩展以上三类构件交互模式。扩展后的 ADL 元模型如图 3 所示。与 xADL2.0 类似，我们方法中提出的 ADL 也包含一组 Schema 来描述软件产品线建模的各个方面。表 1 列出了对这些 Schema 及其元素的详细介绍。

3.3 体系结构描述实例

基于扩展 xADL2.0 的描述语言，我们完成了对图 1 中图书馆系统实例的 DSSA 建模。图 4 是其部分体系结构模型(仅包括 Structure 层上元素)。其中构件 BorrowBookForm 与构件 BookManage 通过连接器交互；提供 Optional 功能“显示图书图片”的接口 Intf_BookImage 通过编织器编织到连接器所包含的查询图书交互上；Optional 资源 ImageContainer 通过 Introduce 关系被插装到目标构件 BorrowBookForm 中。

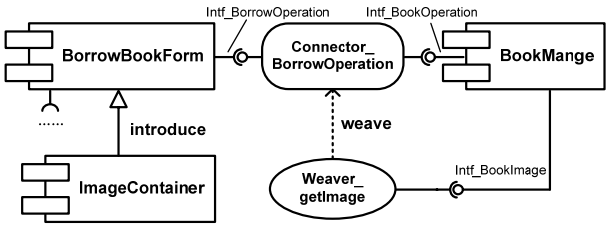

图 4 产品线体系结构模型(部分)

由于 xADL2.0 基于 XML，因此我们能将体系结构以 XML 文本格式记录下来。图 5 是图 4 中体系结构模型的 XML 描述片段。由于篇幅有限，部分元素将被省略。

```
<types:archStructure ...>
  ...
  <types:component types:id="componentffffff85-201a938a-0085a6b6-381d0006"
types:name="BorrowBookForm" xsi:type="types:Component">
    <types:require_interface types:id="interfaceffffff85-201c42f1-912ad1ce-381d0066"
types:name="Intf_BorrowOperation" xsi:type="types:Interface">
      <types:method ...> ... </types:method>
    </types:require_interface>
    ...
  </types:component>
  <types:connector types:id="connectorffffff85-201d524f-a027f377-381d00e1"
types:name="Connector_BorrowOperation" xsi:type="types:Connector">
    <types:service_role xlink:href="#interfaceffffff85-201c42f1-912ad1ce-381d0066"
xsi:type="instance:XMLLink" xlink:type="simple"/>
    <types:request_role xlink:href="#interfaceffffff85-201d0fce-a54d3edd-381d008a"
xsi:type="instance:XMLLink" xlink:type="simple"/>
    <types:method_mapping ...> ... </types:method_mapping>
  </types:connector>
  ...
</types:archStructure>
<options:archWeaver>
  <options:weaver types:id="weaverffffff85-28e79888-087d28a3-aa3a0c6d"
types:name="Weaver_BookImage" xsi:type="options:Weaver">
    <options:interaction xsi:type="options:Interaction">
    <options:connector xlink:href="#connectorffffff85-201d524f-a027f377-381d00e1"
xsi:type="instance:XMLLink" xlink:type="simple"/>
    <options:message xlink:href=... />
  </options:interaction>
  <options:advice xsi:type="options:Advice">
    <options:operator xsi:type="options:WeavingOperator"> after </options:operator>
    <options:interface xlink:href="interfaceffffff85-2049ffed-eacc85dd-381d0257"
xsi:type="instance:XMLLink" xlink:type="simple"/>
    <options:method xlink:href= ... />
    <options:context_mapping xsi:type="options:Context_Mapping">
    ...
      </options:context_mapping>
    </options:advice>
    <options:guard xsi:type="boolguard:BooleanGuard">
      <boolguard:booleanExp xsi:type="boolguard:BooleanExp">
        <boolguard:equals xsi:type="boolguard:Equals">
          <boolguard:symbol
xsi:type="boolguard:Symbol">ShowBookPicture</boolguard:symbol>
          <boolguard:value xsi:type="boolguard:Value">"true"</boolguard:value>
        </boolguard:equals>
      </boolguard:booleanExp>
    </options:guard>
  </options:weaver>
</options:archWeaver>
<introduction:archResource>
  <introduction:resource xsi:id="resourceffffff85-2901e222-67a9f64b-aa3a0e9e"
xsi:name="ImageContainer" xsi:type="introduction:Resource">
    ...
  </introduction:resource>
  <introduction:introduce xsi:id="introduceffffff85-29017d67-38e523d9-aa3a0e79"
xsi:type="introduction:Introduce">
    <introduction:resource
xlink:href="#resourceffffff85-2901e222-67a9f64b-aa3a0e9e"
xsi:type="instance:XMLLink" xlink:type="simple"/>
    <introduction:component
xlink:href="#componentffffff85-201a938a-0085a6b6-381d0006"
xsi:type="instance:XMLLink" xlink:type="simple"/>
    <introduction:guard xsi:type="boolguard:BooleanGuard">
      ...
    </introduction:guard>
  </introduction:introduce>
</introduction:archResource>
```

图 5 产品线体系结构的 XML 描述(片段)

图 5 中 XML 描述分为三个部分。在<types:archStructure>中基于FDSPL Schema描述产品线体系结构的基础部分，包括构件、接口以及连接器元素，尤其在连接器中包含交互构件接口之间的方法映射。在<options:archWeaver>中依据FDOptions Schema描述编织模式的构件交互，主要包括定义Weaver元素的编织点以及需要编织进的方法。另外，由于Weaver具有Optional特性，因此Weaver元素包含BooleanGuard的条件定义。Optional的资源在<introduction:archResource>中被描述，与Weaver元素类似，introduce关系也具有Optional性质。在图4的实例中，由于Optional的功能“图书图片显示”由资源ImageContainer与接口Intf_BookImage提供的功能共同完成，所以在ADL中，它们包含相同的Guard条件，表示这两者具有相同的绑定状态(同时被选定或被移除)。

3.4 ASSA的定制

ASSA是根据应用需求，通过对DSSA中可变点进行定制得来的。在xADL2.0中，每个可变元素的Guard条件是由Symbol，Value以及两者之间比较关系所组成的布尔表达式[7]，例如图5中ShowBookPicture==”true”的Guard条件表示图4中Optional的编织器及资源是否被选定的条件。

在定制过程中，应用开发人员被要求对所有的Symbol根据实际需求设定新的Value值，进而计算Guard表达式的真值，最后得到可变元素的绑定决策。在ADL的描述方面，定制的结果表现为：
(1)Optional的编织器被保留或移除。在保留的情况下编织器不再具有Optional性质(其Guard条件被去除)。
(2)Alternative构件的构件类型指向一个特定变体的构件类型。

应用开发人员有时需要修改定制出的ASSA，例如加入新的构件，修改已有构件接口等。这些修改是被允许的，前提是在ASSA中不能加入新的可变性，同时要为新的构件指定构件类型。

4 基于产品线体系结构的应用产品组装实现

ASSA定义了应用产品实现的蓝图，指导了构件组装的过程。在我们的方法中，构件是组成产品的基本复用单元，其实现体由构件类型指定。负责构件交互的粘合代码(Glue Code)则根据构件交互模式自动生成。因此，应用产品的组装实现过程可分为构件组装配置阶段、粘合代码自动生成阶段以及最后的产品发布阶段。

4.1 基于ASSA的构件组装配置

根据第3节的ADL定义，构件类型的实现体类型可进一步扩展。在我们当前的研究工作中，构件的实现体被定义为Java Package，每个构件包含Package下的一组class与interface。其中有一个class作为控制类，它实现服务接口声明的方法，并且通过请求接口向外界请求服务(但在初始情况下请求接口没有被实现)。

在正式组装开始之前，ASSA的完整性将首先被验证，以确保所有构件(DSSA中Abstract构件除外)的构件类型都指向Java Package。同时每一个Java Package包含的接口类必须与构件定义中的一致。对于DSSA中的Abstract构件，由于仅包含构件接口描述而没有对应的实现体，组装过程将会为应用开发人员自动生成interface与class模版，要求他们实现其中的具体方法。当验证及Abstract构件的实现工作完成之后，所有的构件实现体被拷贝至工作空间。

4.2构件交互模式的自动实现

ASSA 中的连接器元素表示接口连接模式的构件交互，相应的粘合代码将被自动生成为工作空间中的连接器class。该class创建请求角色(request role)关联的服务接口的实例，通过该实例能够调用具体的功能方法。同时 class 实现服务角色(service role)关联的请求接口，在接口的方法实现中，依照连接器元素定义的方法映射(Method Mapping)生成调用代码。另外，由前一小节可知，构件的请求接口在初始情况下未被实现，在我们的组装方法中，将通过反射机制从配置文件中读取具体的接口实现类，而此实现类即为自动生成的连接器 class。此过程的实现示意图如图6所示。

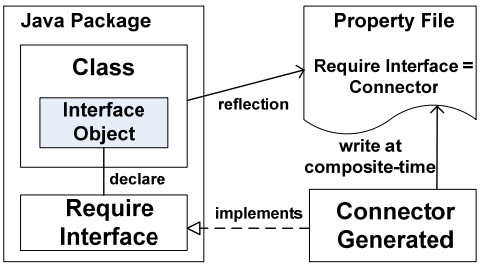

图 6 基于反射的请求接口实现

图 4 体系结构中 BorrowBookForm 构件与 BookManage 构件之间的交互代码在图 7 中列出。连接器类 Connector_BorrowOperation 实现构件 BookForm 的请求接口 Intf_BorrowOperation(行 1)，并且声明与请求角色关联的服务接口 Intf_BookOperation 的对象(行 5)。在构造函数中，使用服务构件的控制类 BookManage 实例化此对象(行 10)。最后，生成调用代码实现 Intf_BorrowOperation 中包含的方法(行 15-24)。

```
1  public class Connector_BorrowOperation implements
                                        BookForm.Intf_BorrowOperation {
2    //Require Component Instance: Reserve the Link
3    BookForm.BookForm requireComponent;
4    //Request_Role(Provide_intf): BookOperation.Intf_BookOperation
5    BookOperation.Intf_BookOperation provide_intf;
6    //Be Weaved: BookOperation.Intf_Image
7    BookOperation.Intf_BookImage weaver_intf;
8    public Connector_BorrowOperation() {
9      //Class Implement Provide_Intf: BookOperation.BookOperation
10     provide_intf = new BookOperation.BookOperation();
11     //Class Implement Weaving Intf: BookOperation.BookOperation
12     weaver_intf = new BookOperation.BookOperation();   }
13   public void reserveLink(BookForm.BookForm bookForm) {
14     requireComponent = bookForm;   }
15   public BookInfo getBook(String bookno) {
16     //Return Value Type: BookInfo
```

```
17      BookInfo BookInfo_value = provide_intf.searchBook(bookno);
18      //Weaving after Interaction
19      weaver_intf.showCoverPicture(requireComponent.getImageContainer(), bookno);
20      return BookInfo_value;    }
21    public boolean reduceBookAmount(String bookno) {
22      //Return Value Type: boolean
23      boolean boolean_value = provide_intf.borrowOneBook(bookno);
24      return boolean_value;    }
25  }
```

图7 接口交互模式与编织模式的粘合代码实现

ASSA中的编织器表示编织模式的构件交互。由于编织模式采取AOP的思想，一般而言应将其实现为AspectJ文件编织到目标程序中。但在我们的方法中，连接器作为编织目标是自动生成的，所以编织模式的实现能够被嵌入到相关的连接器粘合代码中。编织的具体目标是构件接口间的一次交互，因此连接器代码中对应的方法实现将被修改，根据编织目标点(before,after,around)以及上下文映射关系(Context Mapping)调用需要编织进的方法。图7中的代码能够实现接口Intf_BookImage被编织到连接器上的场景。此时，连接器class声明此接口的对象(行7)，并在构造函数中使用接口的实现类BookOperation将其实例化(行12)，然后在编织目标点的接口交互(方法调用)之后调用Intf_BookImage提供的方法(行19)。由于设置图片方法需要使用图片框的宿主类，因此在连接器class中还需要保存对请求服务的构件控制类的引用(行3，13，14)。

Introduce 关系表示引入模式的构件交互。它基于 AOP 的 Introduction 机制，将资源、实例化方法及其它附加代码包装为目标构件的 inter-type [10]，同时根据 Introduce 的插装点(pointcut)定义，被实现为 AspectJ 文件。图 4 实例中图片框 ImageContainer 的引入模式实现代码如图 8 所示。图片框的类型 Canvas(行 3)，实例化代码(行 10-19)以及附加方法(行 20-22)被声明为目标构件的内部类型和方法。另外，将插装点包装为 pointcut，在目标构件的 create 方法之后调用资源的实例化方法(行 4-8)。

```
1   public aspect ImageContainer_Introduce {
2     //Introduce Resource org.eclipse.swt.widgets.Canvas to BookForm.BookForm
3     private org.eclipse.swt.widgets.Canvas BookForm.BookForm.Res_Canvas;
4     //After the create method, do Initialization Method
5     pointcut addCanvas(BookForm.BookForm form) :
6         (execution(* create*(..)))&& this(form);
7     after(BookForm.BookForm form) returning :
8     addCanvas(form) { form.addBookCoverFrame();    }
9     //Resource's Initialization Method
10    public void BookForm.BookForm.addBookCoverFrame() {
11      this.setSize(500, 294);
12      this.Res_Canvas = new org.eclipse.swt.widgets.Canvas(this,
                              org.eclipse.swt.SWT.BORDER);
13      this.Res_Canvas.addPaintListener(new org.eclipse.swt.events.PaintListener() {
14        public void paintControl(final org.eclipse.swt.events.PaintEvent event) {
15          org.eclipse.swt.graphics.Image image =
                  (org.eclipse.swt.graphics.Image) Res_Canvas.getData();
16          if (image != null) {    event.gc.drawImage(image, 10, 10);    }    }    });
17        this.Res_Canvas.setBounds(350, 10, 140, 200);
18        this.Res_Canvas.setData(null);
19        this.Res_Canvas.redraw();     }
20    // Other Introduce Variants and Methods
21    public org.eclipse.swt.widgets.Canvas BookForm.BookForm.getImageContainer(){
22      return this.Res_Canvas;    }
23  }
```

图8 引入模式的代码实现

4.3应用产品发布

此阶段的工作是编译工作空间，生成应用产品。我们使用 AspectJ 编译器[10]进行编译工作(通过调用 ajc 命令)，当工作空间中不存在 AspectJ 文件时，AspectJ 编译与普通的 Java 编译结果相同。

5 原型工具 FdSPLC

本文所提出的方法已被实现成原型工具FdSPLC。它支持软件产品线体系结构建模及应用产品的构件组装实现。FdSPLC是基于Eclipse GMF(Graphic Modeling Framework)[11]的RCP应用系统，它的功能主要包括：

(1)体系结构(DSSA与ASSA)的图形化编辑。编辑元素包括构件、连接器、编织器、资源等。对每个元素可以进一步设置其相关属性，如对构件可以选择对应的构件类型。图9是图书馆管理系统产品线DSSA的编辑界面，主体编辑器支持Structure层次上的建模，左边的视图罗列了Structure以及Type层次上的所有元素。

(2)ASSA的定制。工具提供界面收集DSSA中可变元素的Guard条件，并提示开发人员进行赋值。所有Guard条件被评估后，从DSSA生成应用体系结构。

(3)Abstract构件的模版生成。根据Abstract构件的接口定义，自动实现interface，以及生成class模版。该模版实现服务接口的方法，但方法的实现体需要应用开发人员提供。

(4)应用产品的组装实现。对于ASSA，依照组装流程(第4节)，自动生成粘合代码并且编译工作空间，生产应用产品。

6 结论与展望

软件产品线可变性为体系结构带来更为多样、更加复杂的构件交互情况。除了直接的接口交互之外，还包括更改执行流程的构件交互等。这使得现有的接口连接式体系结构在建模与实现时显露不足。AOP因其“不知觉”性为以上的问题提供了解决方案，它支持在不影响基础程序的情况下将功能编织到目标程序中。因此，我们在基于构件连接式的体系结构描述语言xADL2.0的基础上，结合面向方面的机制，提出了一种软件产品线体系结构建模及构件组装实现方法。方法中的ADL能够清晰描述三类构件交互模式，并且指导构件组装实现的具体过程。

本文提出的方法关注于产品线体系结构建模及构件组装，未涉及到产品线开发的其他阶段。因此，今后的研究工作主要包括：(1)将软件产品线的需求模型(如特征模型)纳入此方法中，试图建立需求模型与基于构件的DSSA之间的映射关系。(2)在构件组装实现阶段，考虑构件的实现版本，引入基于构件的软件产品线配置管理(Configuration Management)。

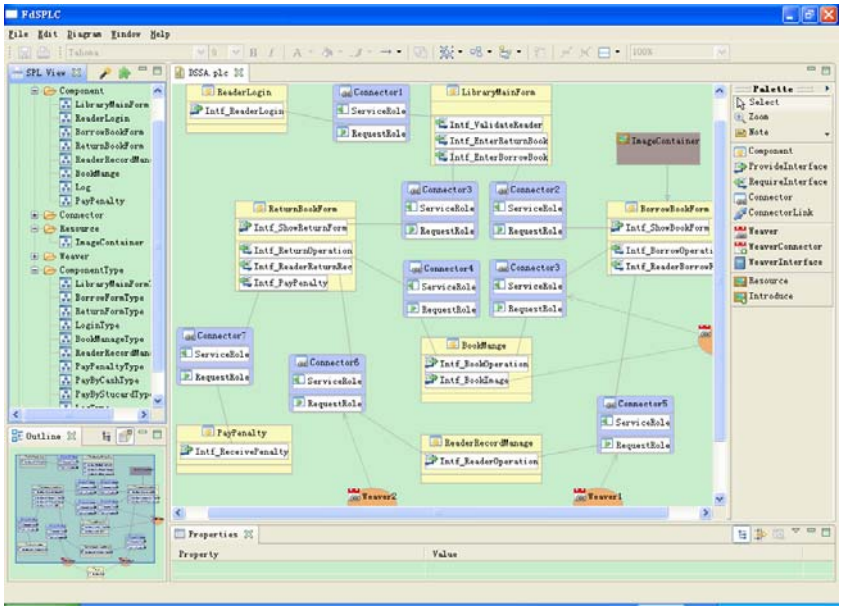

图 9 图书馆管理系统产品线 DSSA 的编辑界面

参考文献：

[1] Show M, Garlan D. Software architecture: perspectives on an emerging discipline [M]. Prentice Hall,1996.

[2] 杨芙清. 软件复用及相关技术 [J]. 计算机科学, 1999, 26(5):1-4.
Fu-qing Yang. Software reuse and related technology [J]. Computer Science, 1999, 26(5):1-4 (in Chinese).

[3] Luckham D, Vera J, Meldal S. Three concepts of system architecture [R]. [Technical Report.CSL-TR-95-674]. Stanford University, 1995.

[4] 张世琨, 张文娟, 常欣, 王立福, 杨芙清. 基于软件体系结构的可复用构件制作和组装 [J]. 软件学报, 2001. 12(9): 1351-1359.
Shi-kun Zhang, Wen-juan Zhang, Xin Chang, Li-fu Wang, Fu-qing Yang. Building and assembling reusable components based on software architecture [J]. Journal of Software, 2001, 12(9):1351-1359 (in Chinese).

[5] Medvidovic N, Taylor R N. A classification and comparison framework for software architecture description languages [J]. IEEE Transaction on Software Engineering, 2000, 26(1), 70-93.

[6] Wei Zhang, Hong Mei, Hai-yan Zhao. Feature-driven requirement dependency analysis and high-level software design[J]. Requirements Engineering,2006,11(3): 205–220.

[7] Eric M. Dashofy, André van der Hoek, Richard N. Taylor. A comprehensive approach for the development of modular software architecture description languages [J]. ACM Transactions on Software Engineering and Methodology (TOSEM), 2005, 14(2): 199-245.

[8] Michalis Anastasopoulos, Dirk Muthig. An Evaluation of Aspect-Oriented Programming as a Product Line Implementation Technology [A]. In Proceedings of the International Conference on Software Reuse (ICSR), 2004 [C]. Springer LNCS 3107: 141-156.

[9] Xin Peng, Li-wei Shen, Wen-yun Zhao. Feature Implementation Modeling based Product Derivation in Software Product Line [A]. In proceedings of the 10th International Conference on Software Reuse (ICSR), 2008 [C]. Springer LNCS 5030: 142-153.

[10] AspectJ Team. AspectJ Project [OL]. http://www.eclipse.org/aspectj/.

[11] GMF(Graphic Modeling Framework) [OL]. http://www.eclipse.org/gmf/.

作者简介：

沈立炜　男，上海人。复旦大学计算机学院博士研究生。主要研究方向为软件产品线、领域工程等。E-mail: 061021062@fudan.edu.cn

彭　鑫　男，湖北黄冈人。复旦大学计算机学院讲师。主要研究方向为构件技术、软件产品线等。E-mail: pengxin@fudan.edu.cn

赵文耘　男，江苏常熟人。复旦大学计算机学院教授，博士生导师。主要研究方向为软件工程、基于构件的软件开发等。E-mail: wyzhao@fudan.edu.cn