基于动态构件框架的构件演化

DCF-Based Component Evolution

符进强　汪　洋　钱乐秋

（复旦大学计算机系　上海200433）

Abstract　This paper presents component evolution in software architecture, proposes a DCF-based evolution method, and describes an instance of the method.

Keywords　Component evolution, Interface, Behavior, Dynamic-Component-Framework(DCF), Dynamic-Plug-Point

1　引言

软件系统是对现实世界模型的反映，现实世界的变化要求软件系统进行相应的演化，软件演化是软件系统的动态行为，贯穿整个软件生命周期，从系统的初始开发阶段到最终的软件维护。软件演化包括软件系统的过程、方法、技术、组织方式等的演化。基于软件构架和构件的开发方法实现的系统比传统的开发方法具有更好的易演化性。软件构架高度抽象地描述了软件系统的结构，包括系统元素的描述、元素之间的交互、用于指导元素复合的模式和这些模式的约束。构件是组成构架的基本元素，是对系统应用功能的实现；构件封装了功能性，有着自己的内部状态信息；构件的实现是异质的（可以用多种语言实现），而且在系统实现中可能使用的是第三方提供的构件。基本构架和构件的系统可以有两种方式的演化，一是整体构架的演化，重组系统的构架，增加、删除系统的构件，修改构件之间的拓扑结构；二是系统中单个构件的演化，着眼于系统的单个构件，修改构件的接口、功能。本文着重于研究构架构件系统中构件的演化，对系统使用的构件进行演化，在不修改系统中其他的构件，保持构件之间的基本连接拓扑结构不变的情况下，将演化后的构件替换系统中的原有构件，实现对系统功能的修改。

2　构件的演化

在基于构架构件的软件系统中，构件作为系统一个特定的功能单位，主要由三部分组成：一组信息、一组行为、一组接口（图1）。信息保存在构件的内部，包括构件内部的状态等，如记录构件被访问次数的状态，构件在实现其功能时将参照这些信息；行为是构件所能实现的功能；接口是构件对外的表现，包括构件对外的属性和方法调用，通过接口，而且只能通过接口才能实现构件与其他构件的交互。

图1　构件组成

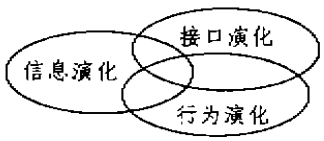

图2　构件演化类型

构件的演化是在现有构件的基础上，对现有的构件进行一定的修改，以满足系统的新需求。从构件组成的角度看，构件的演化可分为三种演化类型：信息演化、行为演化、接口演化（图2）。

接口的演化通过修改构件的接口实现，包括增加、减少、替换原有的构件接口等。单一的接口演化不要求构件内部行为修改，而是修改构件提供的接口，如增加或减少接口参数、类型等，构件行为的演化则是在保持构件对外接口不变的情况下，修改构件内部的具体功能，重新实现构件内部的逻辑。信息演化就是给构件增加新的信息，如构件内部新的状态信息。在大多数情况下，对一个构件的演化不仅仅属于上面提到的三种演化中的一种，而是三种中两种甚至全部三种的联合。接口-信息的联合演化是在构件中增加新的信息，并增加新的接口，以便外部构件能访问该信息；行为-信息的联合演化是在原有构件中增加新的信息，并按照所加的信息修改构件中特定行为，如可在构件中增加一个记录某个接口被访问次数的信息，并按照被访问的次数，相应实现构件中的信息。接口-行为演化是由于修改构件的内部实现，构件可以向外提供新的功能，因此要求接口进行相应的变化。

在对原有构件进行演化时，可能原有的构件已被

符进强　研究生，主攻软件工程。汪　洋　研究生，主攻软件工程。钱乐秋　教授，软件工程。

使用到特定的软件构架中，构件的演化就要尽量不修改系统构架和其他构件，而根据新的需求，实现新的功能。在理想的情况下，演化后的构件与原有的构件相比较应实现：一、相似性，在系统中可以通过与使用原有构件相同的方法使用演化后的构件，新老构件的接口一致。二、保持性，原构件中没有被演化的部分，在演化后的构件中不必进行额外的处理，就可访问。三、开放性、扩展性，演化后的构件应该不仅要实现新的功能，而且能够保持一定的开放性，使得在将来的系统中构件能够进行再次演化。

3 构件演化方式

在构件的开发过程中，往往忽略了构件与外部其他构件之间的接口问题，可能会出现构件之间的接口不一致，解决这个问题的一个最普遍的方法是用包装器(Wrapper)对构件进行包装(图3)。包装器对构件的接口进行封装来适应新的需求环境。包装器的实质是作为一个筛选器，将对原构件的请求进行过滤并调用相应的行为。通过将一个或多个构件作为一个组合构件的组成部分，包装允许构件组合和聚集起来完成新的功能。这种方法的首要优点是将原构件和构件的适配代码分开。

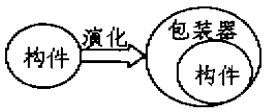

图3 构件包装

图4 继承演化

包装器的使用范围从简单的接口转化到支持一个完全不同的操作集。包装器可以用于组合和聚集，基于这项技术，存在不同的扩展，如接口适应的可配置或可插入适配模块，两种方式的适配模块在一定程度上解决了透明问题。在包装器中新加的代码是不能被重用的，因为它与被包装的构件高度耦合。

包装技术的一种扩展是继承原有构件创建新的构件(图4)。新创建的构件由原构件继承而来，是原构件的子类型。子类型的过程是通过加强原来的构件创建一个新的构件，并重用其中的实现。与包装相似，但是对每一个构件创建新的构件，而不是创建一个包装类型；消息被直接传送到新的构件中。面向对象语言支持通过继承类实现子类。继承可用于重用和代码共享。子类技术使得构件的客户不仅仅能修改接口，而且可以通过函数重载、多态来实现其行为的修改。

包装技术的另一常用的扩展是代理。为了降低包装器和原始构件的耦合度，代理将包装器独立成一个特殊的代理构件，来作为客户和构件的中间构件进行交互(图5)。一个代理在两个构件之间，并用于补偿这两个构件中接口的不一致。代理通过给原构件传送消息，提供对象一级的消息传送。与包装类似，这种演化技术允许对消息进行筛选，并将请求的功能委派到多个构件中。与包装技术不同在于构件不是被嵌入到容器中，它仍然能被直接访问。然而这种技术并没有提供透明，因为客户构件要求与代理进行交互，而不是直接与想要的构件交互，而且在一定程度上对构件的运行效率产生影响。

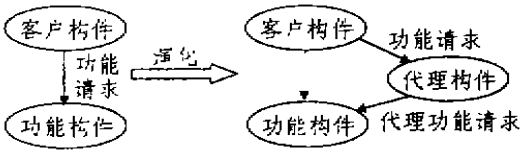

图5 构件代理

如在一个系统构架中，需要通过一个构件执行对数据库的直接访问，现在为了加强系统的安全性，要求所有对数据库的访问都要进行权限设置，对系统进行修改就要修改访问数据库的构件。一个可行的方法就是在请求数据库访问构件与访问数据库构件中间加一权限检查构件作为访问数据库构件的代理。所有访问数据库的请求都将消息发送到新加的代理构件，由代理构件进行权限检查后，将合法的请求再发给数据库构件。

对大部分的系统，在软件的生命周期中，对于构件的演化往往要求构件的接口不变，而修改构件中的内部功能，上面提到的包装技术在解决这样的问题时就比较困难，这是由于在进行构件开发时，没有考虑到这样的问题，特别是使用第三方提供的构件，在没有构件原码的情况下，将很难实现构件内部行为的变动。

在实际的开发过程中，对构件的演化往往不仅是要进行接口演化，而且要在保持构件原有的接口的同时，修改构件内部的部分逻辑。我们在实际的开发过程中，通过不断摸索，构建出一种动态构件框架，将构件的演化机制内置，能对构件内部逻辑进行演化，而保持构件的接口不变。

4 动态构件框架演化

一个构件的接口在构件的演化中扮演了一个重要的角色，在动态构件框架中，除了上文中提到的构件接口外，构件还新加了用以演化的特殊接口。按照新构件中接口的功能，将构件的接口分为两种：一种用于构件的行为，称为行为接口；一种用于构件的行为演化，称为演化接口。在构件设计时，构件的开发人员在构件中合适的地方引入演化机制，将构件的部分接口开放，用于构件演化，由构件的使用者进行构件的演化。构件中

的每个行为接口与构件内部一组方法相关，构件的演化接口被设置成在特定的行为接口被调用时起作用，动态构件框架中，构件设计人员在构件的行为方法中内置了一些必要的动态插入点，并为这些动态插入点定义相关的演化接口，用户在使用构件时，可以通过访问适当的演化接口，为相关的动态插入点定义回调方法(Callback)，增加或替换成用户需要的代码，构件在实施构件行为时，当构件的方法执行到构件内置的插入点，检查用户在演化接口的设置信息，如果用户已经设置了该插入点的相关操作，构件将直接调用用户设置的操作，并在该操作完成后，再根据操作的返回值——true，false，进行判断是否继续执行构件的方法。简单的情况如上面提到的访问数据库构件，在动态构件框架中，可以在访问数据库方法中，在真正的构件功能执行之前，设置动态插入点，用户使用构件提供的演化接口，给该动态插入点定义回调方法，使用定义的回调方法进行用户需要的权限控制。用户定义的回调方法作为构件的外置模块，在构件执行相关的操作时由演化接口管理器实施调用。

动态构件框架中，所有的构件从可演化构件超类继承，可演化构件超类的结构如图6所示。

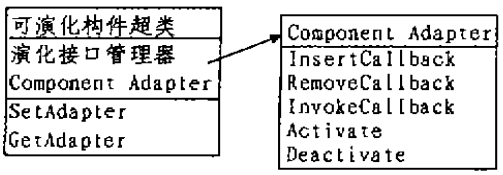

图6 演化构件超类

构件演化接口管理器用于管理和设置各个演化接口的回调方法，主要有三个实现的接口：插入回调方法 InsertCallback(EvoInterface，Callback)，删除回调方法 RemoveCallback(EvoInterface，Callback)，调用回调方法 InvokeCallback(EvoInterface，Callback)。构件管理器的作用是记录用户对每一个可演化的行为接口是否定义了回调的方法，并按照这个信息调用相关的方法。

类 ComponentAdapter 用于管理和设置各个回调方法，并在特定的时候调用。构件的可演化接口需要被实现成：ComponentAdapter getAdapter 和 setAdapter (*ComponentAdapter)，它的任务是连接构件和 ComponentAdapter 中的功能。通过给演化接口设置回调方法，给要调用的构件增加新的代码，与这个演化接口相关的构件行为可进行两个不同行为间演化：排除原有构件的行为，并加上新的行为。两种方式的联合允许应用开发人员直接在接口一级完全重载原构件的行为。

使用回调方法，动态构件框架实现了一个构件内置的演化机制 这种机制的提出有两个方面：一、提供回调方法的基本管理功能；二、定位回调方法调用中要使用到的代码。

构件中要演化的接口被实现在演化接口管理器中，这个类提供了一套 activate(活动)和 deactivate(无效)的演化机制。

构件动态框架机制中，一个构件设计人员可以创建特殊的可演化接口来使构件的重要策略决定可以被演化，通过这种方法，构件的接口增加了，并是开放实现的。动态构件框架的构件演化满足如下的需求：

·嵌入：演化机制在演化过程之前已经内置到构件中。

·着眼点于构架：演化技术是在构架一级的。

·框架独立：能够在不同的构件模型和软件构架风格中实现。

·黑盒：不要求理解构件内部的实现。

·多次演化：演化以后的构件可以进行再次演化。

·一致：演化后的构件保持了一致性，而不是象包装器。

·保持：一旦演化后，构件中非演化的部分仍然可以象演化之前一样直接的访问。

·相似：使用演化后构件的客户不必知道构件的变化。

·处理已有构件的代码：将遗留的构件转化成可演化的构件只需对构件本身有一个很少的理解，而且所要添加的代码也很少。

动态构件框架的演化与包装技术的演化的着重点不同，包装着重于演化构件的接口，动态构件框架的演化则着重于演化构件的行为，演化的机制内置到构件中，构件演化后由原构件与用户增加的行为模块组成组合构件，但在基于动态构件框架的构件开发过程中，开发人员必要考虑到构件中需要演化的决策点，以便将其设定为动态插入点，这给构件的开发增加了难度。

5 动态构件框架演化应用

结合上面动态构件框架，我们将其应用在一个进销存系统中，系统可以针对不断提出的新的变化进行动态演化。

系统主要用于商品入库、销售、出库管理，系统的用户有相对固定的购买群，在系统中保存了客户的信息，而且在一定的限额内，允许客户先提货，后收款。系统中销售、出库的主要流程包括：给购买者开单、出库。首先根据购买者所需的货物进行逐项开单登记，并减少货物的库存，然后购买者再持该单据到仓库提货。考虑到用户对系统中的销售、出库的需求经常会变化，我们对系统中的主要构件：开单、出库构件采用了动态构件框架，使得系统能最大限度地根据用户的具体需求，迅速进行演化以满足需求。

对开单构件和出库构件，我们对用户可能提出的需求变化进行分析，在构件内部内置一些相关的动态插入点，并将与这些动态插入点相关的演化接口开放。

开单构件

```
Component Sale entends Adapter
{
    Interface Sale
    Evolutable Interface Before
    Evolutable Interface After
    Evolutable Interface CheckClient
    Evolutable Interface OpStorage

    Method Sale
  Begin
    Before()
    CheckProductStorage() //检查货物库存
     CheckClient() //检查客户
     SaleToClient() //货物销售
     OpStorage() //对仓库货物库存进行操作
     UpdateClient() //修改客户相关信息
     After()
  End
}
```

出库构件

```
Component ExportStorage entends Adapter
{
    Interface ExportStorage
      Evolutable Interface Before
      Evolutable Interface After
      Evolutable Interface OpStorage

    Method ExportStorage
    Begin
     Before()
     CheckProductStock() //检查货物库存
     ExportStorage() //相关的出库流程
     OpStorage() //对仓库货物库存进行操作
     After()
      End
}
```

在开单构件中，我们在复杂的销售流程中，将检查客户 CheckClient()，和库存操作 OpStorage()定义成可演化方法，并定义了相关的可演化接口。在出库构件中，将出库流程中的库存操作 OpStorage()定义为可演化方法，并提供相关的演化接口。在使用这两个构件时，无须详细地了解构件中复杂的流程，只需使用构件提供的接口。在两个构件中，除了上面提到的功能演化接口外，我们还都增加了两个可演化接口 Before、After，并在构件中内置对应的插入点 Before、After。增加这两个演化的接口，在使用构件时，可演化的程度更高，同样可对这两个演化接口定义相关的操作，实现特定的功能，如可在可演化接口 Before 中，定义一个回调方法来检查开单操作员的权限级别，使构件演化有更大的灵活性。

在系统已经进入正常使用阶段后，用户对系统提出新的需求，要求系统进行几个方面的修改：在给先提货后收款的客户开单时，系统要自动对该客户的欠款限额进行控制；系统中维持销售库存和实际库存，在开单阶段系统将只减少货物的销售库存，而在客户提货时再减少货物的实际库存。根据用户新的需求，我们对系统中的开单构件和出库构件，利用构件提供的演化接口，进行演化。我们使用开单构件的演化接口 CheckClient()，给原有的检查客户方法 CheckClient() 增加新的回调方法，增加构件的外置模块，该回调方法用于检查客户的欠款限额。开单构件在执行开单操作时，构件将自动调用新加的回调方法，并根据该方法的返回值，决定是继续执行余下的开单操作，还是终止开单操作。使用开单构件中的演化接口 OpStorage()，定义新的库存操作方法，并将原有的库存操作方法替换成新的库存操作方法。对出库构件，进行类似的处理，定义满足用户需求的库存操作回调方法，以替换原有方法。当这两个构件中的行为执行到相应位置时，将调用新的回调方法对货物库存进行操作。这样，不改变构件的接口，在保持原有系统构架不变的情况下，对构件实施适当的演化来满足用户的需求。对于演化后的构件，实现了演化的一致性，演化后的构件与原构件比较，除用户增加定义的回调方法外，构件的其他行为保持不变，而且演化后的构件仍然保持了构件中内置的可演化性，用户可继续根据系统需求对构件进行演化。实施演化后的开单构件的框架如图7。

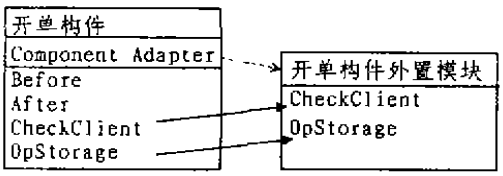

图7　开单构件演化框架图

结论　本文介绍了基于软件构架系统中构件演化的方法，构件演化的主要方式。在动态构件框架中，构件的重要决策利用可演化接口向外开放，可以在不详尽了解构件内部实现的情况下实施构件行为的修改，而且不影响系统构架中的其他构件；构件的演化机制内置在构件内部。

参 考 文 献

1　Bransard F, et al. Toward Software Plug-and-Play. Software Engineering Notes, 1997(3): 19～29

2　Gamma E, et al. Design Patterns: Elements of Reusable Software. Addison-Wesley, Reading MA, 1995

3　Heinenman G T, Ohlenbusch H M. An Evalution of Component Adaptation Techniques: [Technical Report WPI-CS-TR-99-20]. Department of Computer Science, Worcester Polytechnic Institute, Feb. 1999

4　Kemerer C F, Slaughter S. An Empirical Approach to Studying Software Evolution. tEEE Transaction on Software Engineering, 1999, 25(4)

5　Medvidovie N, et al. A Type Theory for Software Architectures: [Technical Report UCI-tCS-98-14]. Department of Information and Computer Science University of California April 1998