基于组合连接器的动态软件体系结构规范方法*)

熊惠民[1,2,3] 应 时[1,2] 虞莉娟[4]

(武汉大学软件工程国家重点实验室 武汉 430072)[1] (武汉大学计算机学院 武汉 430072)[2]

(华中师范大学数学与统计学学院 武汉 430079)[3] (武汉理工大学自动化学院 武汉 430070)[4]

摘 要 动态体系结构的建模与分析是复杂软件体系结构设计的一个重要问题.本文用组合连接器扩展了体系结构描述语言Wright,并由此提出了一种规范动态体系结构的形式化方法。为了支持动态机制,还提出了动态角色的概念。通过实例说明,该方法能将动态体系结构的两种基本形态的描述统一起来,并能为动态软件体系结构设计提供一种增量式的开发方法。由于该方法基于组合的机制,从而适用于体系结构重用。

关键词 组合连接器,动态软件体系结构,动态角色,重用,体系结构描述语言,Wright

Specification of Dynamic Software Architecture Using Composite Connectors

XIONG Hui-Min[1,2,3] YING Shi[1,2] YU Li-Juan[4]

(State Key Laboratory of Software Engineering, Wuhan University, Wuhan 430072)[1]

(Computer School, Wuhan University, Wuhan 430072)[2]

(Department of Mathematics & Statistics, Hua Zhong Normal University, Wuhan 430079)[3]

(School of Automation, Wuhan University of Technology, Wuhan 430070)[4]

Abstract A critical issue for complex software architecture design is the modeling and analysis of dynamic architecture. This paper argues for a formal approach to specification of dynamic software architecture in terms of composite connector based Architecture Deascription Language Wright. To capture dynamic structure. the concept of dynamic role have also been identified. A very simple example is presented to illustrate how they can be used to create dynamic evolution of architecture. As it will be shown that it provides identical way for designing two basic forms of dynamic architecture and an incremental mechanisms for its development, and is avaiable for software architecture reuse owing to the composition mechanism.

Keywords Composite connector, Dynamic software architecture, Dynamic role, Reuse, Architecture description language (ADL), Wright

1 引言

随着软件系统变得越来越复杂,在软件工程界出现了基于组件的软件工程(CBSE)。由于CBSE将一个软件系统看作是由组件所组成的,因此此时如何组织各个组件即软件体系结构就成为一个核心的设计问题。这其中关键的一步就是为描述和分析体系结构提供一种形式化的方法[1,2]。软件体系结构描述语言(ADL, Architecture Description Language)是形式化描述软件体系结构的基本工具,也是对软件体系结构进行求精、验证、演化和分析的基础。近些年来,在ADL方面已经取得了相当大的进展,已出现的ADL包括Wright[3~5], Darwin[6,7], Rapide[8]等。

一般认为,软件体系结构描述的是"一个软件系统的组件结构、组件间的相互关系以及对它们的设计和动态演化进行管理的一些准则"[1]。软件体系结构可能是静态的,也可能是动态的。如果一个软件系统在运行时,组件间的关系和结构不发生变化,则称该软件系统的体系结构是静态的,反之则称为是动态的。在工业实践中,一个复杂软件系统的体系结构经常是动态的,如系统的自举、动态配置、自适应等等。而在某些用于安全或紧急任务的系统里,动态体系结构甚至是必不可少的,如空中交通控制系统。因此,一个ADL必须能提供对动态体系结构的描述与分析。

一个软件体系结构的动态变化可以包含两个方面,即拓扑结构的改变和交互模式的改变。拓扑结构的改变也称结构演化(structural evolution),指的是组件间连接形式的改变以及组件实例集的增减。而交互模式的改变则指的是互连组件间协议的变更。就目前的研究而言,多数动态ADL致力于描述软件体系结构拓扑结构方面的动态变化,而较少关注交互协议的动态变更。

在过去的研究中,作者基于重用的目的,借用CSP进程间的运算,以元操作的形式扩展了Wright语言,从而在Wright语言基础上提出了组合连接器的概念[9]。本文试图基于这一概念提出规范动态体系结构的形式化方法。这样做,一方面可以将体系结构的两种动态行为的描述方式统一起来,从而增强动态ADL的表达能力,也有利于对系统进行推理与分析。另一方面,可以同样支持软件重用和提高软件

*)国家自然科学基金项目(60473066)和湖北省青年杰出人才基金项目(2003ABB004)。**熊惠民** 博士生,讲师,主要研究领域为软件体系结构、软件形式化方法、计算模型;**应 时** 教授、博士生导师,主要研究领域为基于组件的软件工程方法学、软件可重用性与互操作性、软件体系结构和模式;**虞莉娟** 讲师,主要研究领域为电路理论、自动控制理论及应用。

系统的可扩展性，从而为软件设计提供一套增量式或者组合式的开发方法，同时也有利于对系统的理解。

我们提出的形式化方法基于 Garlan 的一个思想。Garlan 认为，一个动态的软件体系结构可以被看作是一组可能的静态体系结构的“快照”(snapshots)，而“快照”之间的相互转换则取决于特定的重配置事件的触发[5]。按照这样的观点，我们认为可以采用下述方案描述一个动态的软件体系结构：首先用静态 ADL 来分别独立地描述动态的软件体系结构所涉及到的各个可能的“快照”，在此基础上，再采用我们定义的连接器组合机制和组合元操作来定义静态体系结构的转换规则。也就是说，复杂的动态体系结构可以看作是由简单的静态体系结构组合而成。

本文余下部分是这样安排的：第 2 节在回顾作者相关的工作后提出了动态角色的概念；第 3 节引出一个实例，并详细论述我们对该实例的两种解决方案；第 4 节论述了与本文有关的工作，并将它们与作者的工作进行了比较；最后总结全文并展望下一步的工作。

2 组合连接器与动态角色

2.1 组合连接器与组合元操作

我们的方法建立在软件体系结构描述语言 Wright 基础之上。Wright 把软件体系结构看作是由组件和连接器构成的框架，组件封装了软件系统的功能信息，而连接器则实现了组件间的通讯。具体地说，Wright 基于事件来规范软件体系结构的行为，并采用组件、连接器、端口、角色等术语来进行刻画，其形式化理论基础是 CSP[10]。CSP 的每个进程定义了一个事件的字母表以及该进程能够呈现的事件发生模式，Wright 则用 CSP 进程来规范组件的计算、组件端口的行为以及连接器的角色行为与粘结。我们认为，Wright 能结构清晰地、显式地表达组件间的交互(即连接器)，分析系统的组合特性(如无死锁性)，易于系统设计者对软件体系结构进行表达、理解和静态检查，是一种理想的静态体系结构描述方法。

为了支持连接器规范的重用，我们提出了一种基于反射机制的连接器组合的形式化方法[9]。该方法按照反射的观点把一个组合连接器规范分为两级：基级和元级。基级用 Wright 来描述各个简单的连接器，元级则描述简单连接器的组合。为了实现连接器组合这种元计算，我们在元级引入了组合元操作。也就是说，组合连接器是指通过组合元操作将多个连接器组合起来，得到一个新的、较为复杂的连接器。

为了规范一个组合连接器，我们依循 Wright 的方法，对它的每一个角色以及角色间的粘结进行规范。由于组合连接器的每个角色以及角色间的粘结都是由简单连接器经过组合元操作得到的，而 Wright 又把角色和粘结都看作进程，因此连接器组合的元操作实质上也可以看作是进程间的运算。我们借用 CSP 进程间的运算，构造了一组连接器组合的元操作，如表 1。

表 1 组合元操作及其所对应的 CSP 运算符

类型	组合元操作	对应 CSP 运算	语义解释
关于角色的元操作	Substitute		一个角色替代另一个角色
	ConcurrencyMerge	$\|\|$	一个角色并行充当多个角色
	AlternativeMerge	$\Box$	一个角色选择充当多个角色
	FollowMerge;	;	一个角色依次充当多个子角色
	InterruptMerge;	Δ	随后续子角色初始事件的发生，顺序中断充当多个角色
	LightningMerge	ζ	随定义的中断事件的发生，顺序中断充当多个角色
关于粘结的元操作	Parallel	$\|\|$	并行多个粘结协议
	Decision	$\sqcap$	不确定选择多个粘结协议
	Choice	$\Box$	选择多个粘结协议
	Interleave	$\|\|\|$	交错搭接多个粘结协议
	Follow	;	顺序执行多个粘结协议
	Interrupt	Δ	随后续粘结初始事件的发生，顺序中断执行多个粘结协议
	Lightning	ζ	随定义的中断事件的发生，顺序中断执行多个粘结协议

关于这组元操作，文[9]中有更为详尽的论述。应该指出的是，与其它元操作不同，LightningMerge 和 Lightning 要引用一些事件作为函数参数。为了能便于组合多个角色或粘结进程，我们规定这两个函数的语法格式为：LightningMerge ($c_1.role_1$, event1, $c_2.role_2$, $event_2$, …, $event_{n-1}$, $c_n.role_n$)，Lightning ($c_1.glue$, $event_1$, $c_2.glue$, $event_2$, …, $event_{n-1}$, $c_n.glue$)，它们简捷地表达了 n 个进程(角色 $c_1.role_1$, $c_2.role_2$, …, $c_n.role_n$ 或粘结 $c_1.glue$, $c_2.glue$, …, $c_n.glue$))随着中断事件 $event_1$, $event_2$, …, $event_{n-1}$ 的发生依次中断地执行。

2.2 动态角色

动态角色的思想源自 Darwin。Darwin 是一种对复杂的、可伸缩的、分布式系统进行组件配置的规范语言。为了捕获软件系统的动态结构，Darwin 提出了两种技术，即惰性实例化和直接动态实例化。惰性实例化通过将一个组件实例声明为惰性，来实现这样的动态特性：只有与之互连的组件请求获得该组件提供的服务时，该组件才进行实例化，从而使得系统结构在一定范围内按照需要动态变化；与惰性实例化不同，直接动态实例化则是将一个组件类型声明为动态特性，再将这个动态组件类型绑定到某个需要动态实例化服务的组件上，从而允许不受限的结构演化[7]。

与上述两种动态技术类似，我们也可以在组合连接器的描述中，将连接器的角色声明为动态特性，从而提出动态角色的概念。由于用组合连接器规范软件体系结构，体系结构的动态性直接表现为连接器交互协议的更替。因此，我们认为，动态角色的动态性指的是，某时刻它与组件端口进行绑定与否，取决于此时组合连接器的粘接是否规范了该角色，即此时的交互协议是否需要该角色参与。若是，则动态角色就要被相应的组件端口实例化；若不是，则动态角色就不被相应的组件端口实例化，如果已经实例化，那么该实例化也要被取消。所以，用组合连接器的动态角色来规范动态体系结构，实质上是要将软件体系结构的动态结构演化用连接器角色的端口实

例化的演化来描述。

3 一个动态体系结构的例子

下面通过一个经典的例子[5]来说明我们的方法。考虑这样一个 C-S 连接器，它表示两个组件 client 和 server 之间的连接协议。根据 Allen 的观点[3]，一个连接器被定义成一组角色(role)规范和一个粘结(glue)规范。角色描述了每个参与交互的组件预期要发生的行为，或者说，它决定了一个组件如若成为角色实例所应承担的责任。而粘结则描述了对这些角色活动的协调与约束。上述连接器类型用体系结构描述语言 Wright 进行规范，如图 1 所示。

```
connector C-S-connector =
  role client = (request! x → result? y → client) □ §
  role server = (invoke? x → return! y? →server) □ §
  glue = (client. request? x → server. invoke! x → server. return? y
      → client. result! y → ? glue) □ §
```

图 1 Wright 规范：C-S-connector

在上述的简单情形中，我们假定了 server 总能可靠运行。但现实情况并非如此，server 经常要运行在不稳定的环境里，这种不稳定性包括通讯线路的不稳定以及 server 自身由于遇到不可预知的情况而出现的运行故障。因此我们必须构造一种动态的软件体系结构以支持系统的健壮运行。针对这两种情形，通常有两种解决方案[5]：一是当系统遇到通讯线路不稳定时，动态地改变 client 与 server 的通讯方式，使得 server 每次得到 client 的一个请求时都给它发出一个确认收到的消息，而若无反馈消息 client 则重发请求(为了简单起见，这里假设 client 总能收到 server 的反馈消息，并且仅有 client 的请求在通讯过程中才可能被丢失，其他复杂的情形可以类似考虑)；二是我们再提供一个备用的 server，当主 server 出现故障时，系统自动连接到备用的 server 上。

3.1 方案 1 的形式化规范

采用方案 1 的软件系统，它的体系结构的动态性表现为组件间的交互协议要随环境变化而进行相应的变更。当通讯线路运行稳定时，系统就采用简单而高效的交互协议；而一旦通讯线路变得不稳定，系统则用低效但可靠的交互协议来替换。因此，这里的软件体系结构应该包含了两个连接器：一个连接器用来规范通讯线路运行稳定时的情形，它可以重用图 1 中的 C-S-connector 连接器；另一个连接器用来规范通讯线路运行不稳定时的情形，其描述如图 2 所示。

```
connector Cc-Sc-connector =
  role client = (request! x → ((confirm → result? y → client)
→□client ) □ §
  role server = (invoke? x → confirm → return! y → server) □ §
  glue =client. request? x → (glue□ server. invoke! x →
server. confirm → client. confirm → server. return? y →
client. result! y → glue)? □ §
```

图 2 Wright 规范：Cc-Sc-connector

在上述连接器规范中，我们用事件 confirm 表示 client 与 server 之间进行的通讯反馈，它的发生代表 server 确认收到了 client 的请求。

C-S-connector 连接器和 Cc-Sc-connector 连接器代表了软件体系结构的两种状态，系统要在这两个状态间进行转换，从而获得体系结构的“动态性”。因此，下面还需要定义这两个状态之间的转换规则。我们认为，这种转换规则可以用组合连接器来表达，其描述如图 3 所示。

```
connector C1-S1-connector
  composite: cs1: C-S-connector
             cs2: Cc-Sc-connector
  role client = LightningMerge(cs1. client, event1, cs2. client,
event2, client)
  role server =LightningMerge(cs1. server, event1, cs2. server,
event2, server)
  glue = Lightning(cs1. glue, event1, cs2. glue, event2, glue)
```

图 3 方案 1 的形式规范：C1-S1-connector 连接器

该形式规范首先通过保留字 composite 标明 C2-S-connector 是一个组合连接器，并且该组合连接器是用两个不同类型(即 C-S-connector 和 Cc-Sc-connector)的连接器组合而成的。

该形式规范接着描述了组合连接器包含的两个角色。角色 client 描述了该交互协议中 client 的行为，角色 server 则描述了 server 的行为。从直觉上讲，组合连接器 client 角色(或 server 角色)某时要充当 cs1. client (或 cs1. server)的角色，某时又要充当 cs2. client (或 cs2. server)的角色，当然每次只能“扮演”多个角色中的一个。我们通过 LightningMerge 函数实现了用一个角色充当多个角色的目的，并且定义角色间的转换要取决于事件 event1 与 event2 的发生。这里 event1，event2 是与通讯线路状态相关的两个中断事件，如它们可以这样定义：当某两个通讯事件发生的间隔多于一个较长的时间时，event1 发生；而当少于一个较短的时间时，event2 发生。

该形式规范最后描述了对上述两个角色的约束与协调——粘结 glue。它通过函数 Lightning 将两个子粘结进程 cs1. glue 与 cs2. glue 顺序中断地组合起来。

需要说明的是，在对组合连接器的角色和粘结进行规范的过程中，函数 LightningMerge 和 Lightning 都采用了递归结构，从而实现了角色和协议周期性的替代与更迭。

3.2 方案 2 的形式化规范

采用方案 2 的软件系统，它的体系结构的动态性表现为其拓扑结构方面的动态变化。在该方案中，C-S 系统有一个 client 和两个 server。但不论何时，系统只包含 client 与某个 server 的连接，即两个 server 要随主 server 运行情况的变化交替地为 client 提供服务。显然，client 与每个 server 的通讯协议仍然是 C-S-connector，从而可以重用图 1 的连接器。但由于任何时刻 client 只能与一个 server 进行交互，因此两个 server 的运行还必须进行相互协调。这种协调即代表了系统在体系结构的两种状态之间进行转换。我们认为，体系结构这种转换机制也同样能用组合连接器来表达，其描述如图 4 所示。

```
connector C1-S2-connector
  composite: cs1: C-S-connector
             cs2: C-S-connector
  role client =LightningMerge(cs1. client, event1, cs2. client,
event2, client)
  dyn role flakyserver = Substitute(cs1. server)
  dyn role slowserver = Substitute(cs2. server)
  glue = Lightning (cs1. glue, event1, cs2. glue, event2, glue)
```

图 4 方案 2 的形式规范：C1-S2-connector 连接器

该形式规范首先说明 C1-S2-connector 是一个组合连接器。具体地说，它是通过两个具有同一类型——C-S-connector 的连接器组合得到的。

该形式规范接着描述了组合连接器包含的三个角色。

角色 client 描述了该交互协议中 client 的行为。从直觉上讲，组合连接器 client 角色所代表的组件端口既可以调用 cs1. server，又可以调用 cs2. server. 即它既要能充当 cs1. cli-

ent 的角色，又要能充当 cs2. client 的角色。我们通过 LightningMerge 函数实现了用一个角色充当多个角色的目的，使得 client 将随 event1 或 event2 的发生，交替"扮演"两个角色中的一个。这里 event1 和 event2 是与主 server 运行状况有关的两个中断事件，如它们可由主 server 的运行监控程序产生。同样地，为了实现角色周期性的更迭，这里也用到了递归结构。

角色 flakyserver 描述了该交互协议中主 server 的行为，角色 slowserver 则描述了从 server 的行为。函数 Substitute 表明，这两个角色实际上分别是其子连接器 cs1 和 cs2 的 server 角色。从而，flakyserver 具有与 cs1. server 相同的行为规范，slowserver 具有与 cs2. server 相同的行为规范。

注意到，为了表达拓扑结构的动态性，角色 flakyserver 与角色 slowserver 都被定义成了动态角色。也就是说，它们只有在组合连接器的粘结 glue 需要的时候才被组件端口实例化。这点与我们的直觉是相符的。

该形式规范最后描述了对上述三个角色的约束与协调——粘结 glue。通过函数 Lightning 将两个粘结进程 cs1. glue 与 cs2. glue 顺序中断地组合起来，实现了两种通讯协议的动态过渡。这里为了允许描述一个潜在无限交替的结构，我们在函数 Lightning 中用到了递归。

4 相关工作比较

并不是所有的 ADL 都支持动态体系结构的建模。比较典型的动态 ADL 有 Wright，Darwin，Rapide 等。

动态 Wright 用一个外来的实体 configuror 来显式地建模系统的动态演化行为。Configuror 知道系统的所有体系结构细节(包括所有的组件、连接器和连接)，它的职责是去管理和修改系统的体系结构，它用一种类似面向对象语言的方法来动态生成组件，从而获得动态的软件拓扑结构[5]。显然，这种解决方案对分布式系统是不合适的。本文用静态 Wright 来描述体系结构的各个状态，再用组合连接器来表达体系结构的转换。由于连接器本身是一个局部的概念，而组合连接器也只是将相关的连接器连接起来，它的职责是去管理系统与之相对应的那一部分体系结构的动态变化，因此这里的方法能适用于分布式环境。

Darwin 是为了规范分布式系统的体系结构而提出的一种声明式绑定语言。它仅仅关心体系结构的拓扑结构方面，希望为结构化一个系统的行为规范提供框架。Darwin 并不显式地建模连接器，在其中组件是通过获得服务来进行交互，组件间的每个交互用一个关于服务的需求与提供的绑定(binding)来表示[6]。因此，这种语言并没有涉及本文所关注的动态体系结构中系统的重配置行为与组件的计算行为之间的交互问题。

与 Wright 类似，Rapide 也用一种类似面向对象语言的方法来动态生成组件。但它一般不能确定在其执行过程中将要生成哪些拓扑结构[8]。因此，Rapide 关注的是关于系统执行迹的集合的模拟与分析，而我们关注的是体系结构的静态分析与检查。

也有人使用反射的概念来研究动态软件体系结构。Cazzola 在大规模体系结构编程(APIL)的研究中将反射概念引入到软件体系结构中，而提出了体系结构反射的概念。并与计算反射相对应将体系结构反射分为两种，即拓扑反射(topological reflection，与结构反射类似)和策略反射(strategic reflection，与行为反射类似)[11]。APIL 在词法上严格区分元级和基级组件，也区分组件和连接器，这与我们的形式化方法类似。而与 APIL 不同的是，我们提出的形式化方法着眼于软件体系结构的规范，而不是体系结构与实现之间的因果联系。

目前已出现了一些带反射特性的 ADL，如 Pilar，ArchWare 等。Cuesta 等提出了一种通用的反射概念框架 MARMOL，并基于该模型而构造了一种反射性体系结构描述语言 Pilar。Pilar 以 CCS 为其形式化理论基础并为实现反射进行了扩展，它定义的反射原语包括 Avatar(α)，Reify(ρ)，Destroy(δ)等[12]。与我们不同的是，在 Pilar 中，只有组件一种语法实体，连接器被当作元组件来看待，而对组件与元组件的描述则是统一的，它们有相同的形式。ArchWare 项目组则希望将软件体系结构与反射系统的研究进行整合，为基于体系结构的软件开发方法提供一套框架、语言和工具。为了规范动态的体系结构，它提出的描述语言 ArchWare 组合了多个概念：用来规范可执行体系结构的 π 演算、用作系统自省的 hypercode、以及一个能增量式地拆散运行系统的分解算子和用于新建和绑定组件的结构反射[13]。ArchWare 的目的是要提供一种可执行的体系结构规范。与 ArchWare 不同，我们是基于重用的目的来研究动态体系结构，希望为软件开发提供一种增量式或组合式的设计方法。

就形式化基础而言，Wright，Darwin 和 Rapide 分别以 CSP，pi 演算和 Poset 集作为其数学基础。另外，也可以采用术语重写系统(term rewriting systems)来建模和分析动态体系结构。例如，Inverardi 和 Wolf 就基于一种通用的术语重写系统——化学抽象机(CHAM)进行了研究[14]，他们的方法能够描述体系结构的任意的重配置行为，具有相当强的表达能力。但是，由于用它描述系统必须采用重写规则进行编码，从而这种方法远离了系统设计者的直觉。相反地，我们的方法则能够直接地、显式地表达设计者处理重配置的想法。

还可以用图文法(graph grammars)来建模和分析动态体系结构中可被允许的拓扑结构[15]。虽然图文法能够很好地捕获系统转换的模式，但是到目前为止这种方法还不能将体系结构的重配置与系统的行为方面联系起来进行研究，从而它不能如本文一样回答诸如这样的问题：系统什么时候将产生合法的体系结构重配置行为等。

结论以及将来的工作 动态体系结构的建模与分析是复杂软件系统设计的一个重要问题。本文基于重用的目的，提出了一种规范动态体系结构的组合方法。该方法能将体系结构的重配置与系统的行为方面联系起来进行研究，并可允许软件设计者用统一的方式来定义动态体系结构的两种形态，即拓扑结构和交互模式的动态演化。该方法也能适用于分布式环境。

为了获得一个动态体系结构，我们首先用 Wright 的方法规范该动态体系结构的所有可能的"快照"，继而再利用连接器的组合机制描述它们之间的转换。这进一步论证了 Wright 语言以及建立在其上的组合连接器概念表达体系结构的有效性。同时，由于组合方法的采用，动态体系结构规范很好地保持了 Wright 语言可读性强的特点。

为了规范动态体系结构，我们提出了动态角色的概念，我们下一步需要研究的是如何给出动态角色的形式语义以及如

(下转第 265 页)

步调用方式，在这种调用方式中，客户端发出请求之后就不再关心服务端的情况，包括是否执行成功，返回值是什么等等；延迟响应方式是指客户端在发出调用请求之后继续执行，但是经过一段时间之后，客户端再调用相应的方法去检索返回结果，并通过参数指定如何根据调用的结果而执行进一步动作；请求回调方式与延迟响应方式类似，也能得到服务端的响应，但是不同的是这个响应是由服务端通过回调方式来触发的，而不像延迟响应方式由客户端来主动检索的。

结束语 在传统的软件开发领域，目前构件技术如 EJB、CORBA 和 COM 等已经获得巨大的成功，成为软件开发的主流技术。因此，要想使得 Web 服务体系结构的研究成果得到广泛的领域应用，则必须和构件技术相结合。在现有的构件模型中，构件接口描述仅仅是接口的基调，如何扩充构件接口使其包含有关 Web 服务方面的信息是 Web 服务体系结构和构件技术相结合的有效途径。

本文基于 SOA 特点探讨了一种 Web 服务的领域系统构造过程，对其 WS-DSARD 的主要元元素角色、操作、服务构件和服务构件类等进行了定义和较为详细的描述，并从服务构件交互与集成的角度分析了服务构件的组合语义。WS-DSARD 的概念模型具有以下特点：(1)使用结构模型、框架模型、动态模型和过程模型集成刻化面向服务的领域软件体系结构；(2)支持捕获服务作为一组角色间的交互模式，并把每一个角色模拟一个具体构件在参与某个服务实现时所必须展示的行为；(3)把服务视为首要的类建模元素，支持大粒度的 Web 服务构件重用；(4)支持基于 Web 服务构件软件的增量、迭代式开发。

目前，我们已在 WS-DSARD 的概念模型基础上，提出了一种新的基于 XML 的用来描述服务软件体系结构的体系结构描述语言 WS-XADL，并已初步实现了 QoS 驱动的 Web 服务有效发现中间件框架、以及特定领域软件协同集成框架等支撑工具。至于如何更加全面地形式化的描述 WS-DSARD 的各种模型，并基于该模型实现其集成支持环境，还需要在进一步的工作中加以研究。

参 考 文 献

1 Selby R W. Enabling reuse-based software development of large-scale systems. IEEE Transactions on Software Engineering, 2005, 31(6):495～510

2 Papazoglou M P. Service-oriented computing: concepts, characteristics and directions. In: Proceedings of Fourth International Conference on Web Information Systems Engineering (WISE 2003). Los Alamitos, CA, USA: IEEE Computer Society Press, 2003. 3～12

3 Nadhan E G. Service-oriented architecture: implementation challenges. The Architecture Journal, 2004, 2(4):24～32

4 Yang J. Web service componentization. Communications of the ACM, 2003, 46(10):35～40

5 Kang K C. Feature-oriented development of applications for a domain. In: Proceedings of the Fifth International Conference on Software Reuse. New York, NY, USA: IEEE Press, 1998. 354～355

6 Feng X, Guoyuan L, Hao H, et al. Role-based access control system for web services. In: Proceedings of the 4th International Conference on Computer and Information Technology. Los Alamitos, CA, USA: IEEE Computer Society Press, 2004. 357～362

7 Zhao L P, Kendall E. Role modelling for component design. In: Proceedings of the Proceedings of the 33rd International Conference on Technology of Object-Oriented Languages. New York, NY, USA: IEEE Press, 2000. 312～323

8 Rumbaugh J, Jacobson I, Booch G. The Unified Modeling Language Reference Manual, Second Edition. Boston, MA, USA: Addison-Wesley, 2004

9 Rahm E, Bernstein P A. A survey of approaches to automatic schema matching. The VLDB Journal, 2001, 10(4):334～350

(上接第 253 页)

何借助形式语义对系统进行检查与分析。动态角色的概念源于 Darwin 的两种动态机制，借助 pi 演算目前已获得了 Darwin 的形式语义[16]。而在我们过去的研究中，也曾给出了组合连接器的 CSP 语义。因此，将 CSP 与 pi 演算结合起来研究动态角色的语义可能是一种选择。

另一个值得研究的重要问题是关于无限动态结构的描述。所谓无限动态结构，指的是在动态体系结构的结构演化中，系统的拓扑结构能够按某种模式无限地扩展，比如 Darwin 就曾给出了一个基于管道-过滤器体系结构风格的例子[7]。显然为了描述这种结构必须采用递归机制，但是否还应该扩充已有的方法，这需要作更深入的探讨。

参 考 文 献

1 Garlan D, Perry D E. Introduction to the special issue on software architecture. IEEE Trans on Software Engineering, 1995,21(4): 269～274

2 Medvidovic N, Taylor R N. A classification and comparison framework for software architecture description languages. IEEE Trans on Software Engineering, 2000,26(1):70～93

3 Allen R J, Garlan D. A Formal Basis for Architectural Connection. ACM Transactions on Software Engineering and Methodology, 1997,6(3):213～249

4 Allen R J. A formal approach to software architecture: [Ph. D. Thesis]. Pittsburgh: Carnegie Mellon University, 1997

5 Allen R, Douence R, Garlan D. Specifying and analyzing dynamic software architectures. In: Astesiano E, ed. Proc of the 1st Int'l Conf on Fundamental Approaches to Software Engineering. Berlin: Springer-Verlag, 1998. 21～37

6 Magee J, Dulay N, Eisenbach S, et al. Specifying distributed software architectures. In: Schafer W, Botella P, eds. Proc of the 5th European Software Engineering Conf. Berlin: Springer-Verlag, 1995. 137～153

7 Magee J, Kramer J. Dynamic structure in software architectures. In: Kaiser G E, ed. Proc. of the 4th ACM SIGSOFT Symp. on Foundations of Software Engineering. New York: ACM Press, 1996. 3～14

8 Luckham D C, Augustin L M, Kenney J J, et al. Specification and analysis of system architecture using rapide. IEEE Trans on Software Engineering, 1995,21(4):336～355

9 熊惠民，应时，虞莉娟，等. 基于反射的连接器组合重用方法. 软件学报，2006,17(6):1298～1306

10 Hoare C A R. Communicating Sequential Processes. Englewood Cliffs, New Jersey: Prentice Hall, 1985

11 Cazzola W, Savigni A, Sosio A, et al. Architectural reflection: Concepts, design, and evaluation: [Technical Report]. Milano: University of Milano Bicocca, 1999

12 Cuesta C E, de la Fuente P, Barrio-Solorzano M. Dynamic coordination architecture through the use of reflection. In: Lamont G B, ed. Proc. of the 2001 ACM Symp. on Applied Computing. New York: ACM Press, 2001. 134～140

13 Morrison R, Kirby G, Balasubramaniam D, et al. Support for evolving software architectures in the ArchWare ADL. In: Proc. of the 4th Working IEEEE/IFIP Conf on Software Architecture. Washington D C: IEEE Computer Society Press, 2004. 69～78

14 Inverardi P, Wolf A. Formal specification and analysis of software architectures using the chemical abstract machine model. IEEE Trans on Software Engineering, 1995,21(4): 373～386

15 Métayer D L. Describing software architecture styles using graph grammars. IEEE Trans on Software Engineering, 1998,24(7): 521～533

16 Eisenbach S, Paterson R. π-Calculus semantics for the concurrent configuration language Darwin. In: Proc. of the 26th Annal Hawaii Int'l Conf on System Sciences, Volume 2. Washington D C: IEEE Computer Society Press, 1993. 456～462