第 34 卷第 3 期 通 信 学 报 Vol.34 No. 3
2013 年 3 月 Journal on Communications March 2013
doi:10.3969/j.issn.1000-436x.2013.03.005

支持网络动态重构的可信构件模型

袁博[1]，汪斌强[1]，马东超[2]
(1. 国家数字交换系统工程技术研究中心，河南 郑州 450002；2. 北方工业大学 信息工程学院，北京 100041)

摘 要：基于构件接口分离原则和动态软件体系结构技术，提出了一种支持网络动态重构的可信构件模型，对构件构造和组装进行了分析，并提出了可信数据安全封装机制。基于感知、决策和执行分离机制，构件模型可以支持构件重构，通过容器对构件数据进行安全封装，构件模型可以检测和阻止恶意构件的非法操作，保护构件间的数据安全传递。实现了支持可信构件模型的容器原型，实验表明，使用该模型可进行构件连接拓扑关系的重构，可及时发现恶意构件并暂停构件运行，提高了节点的可信性。

关键词：可重构柔性网络；可信；构件模型；安全封装；可重构
中图分类号：TP393 **文献标识码：**A **文章编号：**1000-436X(2013)03-0032-12

Trustworthy component model supporting networks reconfiguration

YUAN Bo[1], WANG Bin-qiang[1], MA Dong-chao[2]
(1. National Digital Switching System Engineering & Techno y Center, Zhengzhou 450002, China;
2. College of Information Engineering, North China University of Technology, Beijing 100041, China)

Abstract: Based on principles of the separation of concerns and the dynamic software architecture (DSA) technology, a component model named TCM (trustworthy component model) was proposed. Formation and assemblage of components were analyzed, then a trustworthy data security sealing mechanism was proposed. By sealed the data, the TCM not only support reconfiguring components, but also prevent component hostility operation, which can destroy the data transmission. A container prototype was implemented, and experimental applications were implemented to validate this approach. The results show that this model can change the topology of components and find out the hostility components, which improve the trustworthy of reconfigurable flexible network node.

Key words: reconfigurable flexible networks; trustworthy; component model; security sealing; reconfiguration

1 引言

随着网络规模不断扩大，大量的个性化业务应用向规模化应用转化，传统的网络体系结构已不能很好地适应这种发展趋势。可重构柔性网络是解决此类问题的一个重要方法，它的理论根源在于动态地改变网络的服务能力使之追随用户业务需求的变化，通过将网络基础设施的物理资源服务能力和服务提供商的业务承载能力相分离，以构建可重构服务承载网的形式，快速、灵活和高效地为用户提供网络服务。物理网络为可重构服务承载网提供网络资源，不同的可重构服务承载网具有不同的服务能力，从而在共享由不同基础设施提供商提供的底层物理网络资源的基础上，能同时支持多个服务特征不同的异质网络体系结构并存，为用户提供多样化的网络服务[1,2]。这种网络得以实现的关键是将底层网络服务与用户业务松耦合，网络节点的软硬件资源也需要被合理的分割。节点设备构件化便是上述机制的一种物理映射，为此需要一种支持重构的构件模型。因为软件和硬件在重构机制的实现上存

收稿日期：2011-11-29；**修回日期：**2012-05-17
基金项目：国家高技术研究发展计划(“863”计划)基金资助项目(2008AA01A323，2009AA01A334)；国家重点基础研究发展计划(“973”计划)基金资助项目(2012CB315900)
Foundation Items: The National High Technology Research and Development of China (863 Program) (2008AA01A323, 2009AA01A334); The National Basic Research Program of China (973 Program) (2012CB315900)

在区别，本文主要研究的是节点设备控制平面软件领域的构件模型。

传统的构件模型设计时考虑的软件运行环境是静态的，即使考虑到运行环境变化也是基于对未来的预测，不能够支持可重构环境中大量未知构件的动态植入和卸载。要支持构件的灵活重构，软件工程层面应该提供一种便于构件重构的构件模型和管理机制，构件模型应该具有感知、决策和执行的能力。即在感知到运行环境或构件发生变化后，构件模型做出相应决策，进而执行方法调用、参数改变或者结构调整等动作。可重构环境中构件的动态植入和卸载会带来很多安全方面的不确定性因素，为保证软件系统的安全性，构件模型还必须为外来构件提供可信交互机制，对构件交互信息进行可信封装，保证任意一个构件的行为和数据不会影响到其他构件和系统环境。在构件重构后还应保证新结构下构件间的可信交互。

针对上述挑战，本文提出了一种可信构件模型(TCM, trustworthy component model)。TCM 包括构件代理、容器和行为构件，使用构件代理(CA, component agent)可完成决策和重构部署，利用 AC(agent client)可完成构件间的可信交互。在构件组装层面引入具有感知功能的容器可隔离底层操作系统的影响，从而实现感知、决策和执行这 3 个功能的独立表达和封装，同时在容器中实现构件数据的可信封装。在此基础上，TCM 使用动态软件体系结构(DSA, dynamic software architecture)技术[3]实现构件连接关系的重构。上述机制一方面使第三方构件可以通过 AC 实现与软件体系结构的松耦合而被灵活地植入和卸载，另一方面构件代理可以对模型下的构件进行管理，保证软件构件系统的灵活重构和可信。

2　相关研究工作

构件的插拔与替换是基于构件的软件工程(CBSE, component based software engineering)中的基本手段之一。在基于构件的软件设计阶段，设计的逐步求精可视为抽象度高的构件不断被更细化的构件替换的过程；在构件的获取阶段，构件的检索可视为检索者所需的构件由构件库中的具体构件所替换的过程；在构件的组装阶段，通过构件的替换可支持系统的灵活重构；在软件运行阶段，构件替换是支持系统升级、维护和演化的主要手段[4]。近年来兴起的网构软件，可通过服务构件的动态替换和编排机制实现网上服务的按需计算，呈现更强的动态性和演化性[5]。构件的替换已成为 CBSE 领域近年来研究的核心和热点问题之一，但是目前还未提出可以支持构件灵活重构的构件模型。

软件适应领域[6]的研究者已经认识到构件接口的功能分离原则在软件适应能力调整中的重要性。较为常见的实现方法是使用策略、规则等手段来单独描述软件适应的决策逻辑，然后通过对决策逻辑的更新来调整适应能力。文献[7]给出了一个实现软件自适应的规则模型。Chisel[8]面向完全不可预期的动态变化，使用策略来描述适应逻辑，通过对策略的动态更新来为软件添加新的适应能力；K-Component 通过适应契约描述语言(ACDL)来描述适应逻辑，实现体系结构在线变化过程中的功能点分离[9]；MADAM 区分功能实现和适应性行为使能 2 个功能点，通过构件的附加属性、效用函数等来支持软件的适应动作[10]。上述方法均只强调决策逻辑从软件功能实现中分离，而本文强调感知、决策、执行三者功能的分离和在线重构，如引言中所做分析，感知能力的独立表达和在线重构对于构件适应能力的调整是十分必要的。此外，上述方法仅强调功能分离，没有考虑开发时带来的设计和编程实现代价，本文则进一步给出了构件模型和使用 DSA 技术来实现构件灵活重构的方法。

与本文工作密切相关的另一研究是基于体系结构的软件自适应。早在 1999 年文献[11]提出了一种基于体系结构的软件自适应模型，该模型由描述适应过程的适应管理和实现体系结构在线变化的演化管理 2 部分组成。除了前面已提及的 K-Component，MADAM 等模型外，Rainbow 研究了软件体系结构在线修改过程中的软件重用问题[12]；FORMAware 强调对体系结构重配置时需要保证功能的完整性和遵守体系结构的约束[13]；Fractal 构件模型要求在设计时就要考虑体系结构配置的变化[14]等。网构软件运行平台[15]也是基于软件体系结构来支持软件的自适应调整，通过自主构件来实现构件内部从感知到行为执行的功能适应[16]。文献[17]提出了一种以“环境信息显式化、互动方式层次化、体系结构可演化”为特征的环境驱动模型。上述工作研究了基于软件体系结构的模型，通过可演化的体系结构来支持软件自适应调整。但是这种调整能力还是依

赖软件体系结构的设计，这种设计方式在封闭的软件架构中是有效的，一旦应用到开放的环境中就暴露出体系结构过于紧耦合的缺点。而本文工作则关注如何在运行时实现软件构件的在线重构。例如构件间拓扑关系改变和通过增加或替换构件来引入新的功能等。本文提出的构件模型，将决策和执行功能交予模型框架实现，构件只专注功能的实现，减少了第三方开发构件的难度，有利于软件在开放异构环境中的应用。

对于运行大量外来构件的软件系统，如何保证软件系统的安全和外来构件的可信是必须要解决的问题。第三方构件主要依靠数据的交互影响其他构件，因此保证构件数据的安全是一种提高构件模型可信性的解决方案。数据的安全封装最早出现在虚拟机研究中[18]。Tal Garfinkel 和 Ben Pfaff 等提出了一种称为 Terra 的可信计算体系结构[19]，它的目标是支持具有不同安全需求的应用同时运行在同一硬件平台之上。Terra 利用虚拟机技术在一个硬件平台上构建多个具有不同安全特性的可信计算系统。通过严格保证虚拟机间的隔离，使得具有较低安全级别的“开放式”系统和具有较高安全级别的“封闭式”计算机系统能够并发地执行在同一硬件之上。但是 Terra 并没有使用安全硬件，而且只为平台上的各种应用提供基本安全保障，并没有建立高等级的信任保证，也没有采用 KVM[20]或 VaxVMM[21,22]的安全共享方法。IBM 公司的 vTPM 系统[23]也是基于 Xen 实现，它能够利用底层硬件的 TPM 模块为上层的多个虚拟机提供可信平台模块服务，从而实现了信息流在虚拟机监视器中的安全传递。HP 也在 TPM 虚拟化方面做了类似的研究工作[24]。但是这些技术面对的是应用复杂多变和易遭受攻击的计算机系统，对于可重构网络中的节点设备，可以利用上述研究的数据安全隔离或封装的思想，不需要开发独立的可信计算平台。

与前人不同，本文提出了一种支持网络重构的可信构件模型，并从构件数据的可信性出发，对构件交互的数据进行安全封装，阻止构件进行恶意操作，提高了整个构件模型的可信性。本文创新点在于提出了一种支持重构的可信构件模型，并阐述了模型的重构机制，组装机制和可信机制，除了应用于可重构柔性网络的节点中，任何需要软件自适应调整的场合均可使用本文提出的构件模型。

3 TCM 构件模型

一般说来，软件构件的结构模型决定了构件的基本形态，它是软件方法学的核心。为建立面向可重构环境的软件构件模型，本节针对经典软件结构模型存在的“预定的环境、紧耦合的结构、集中式的开发和重编程的应变方式”等问题，提出以“松耦合结构、分布式聚合、动态化重构、可信化保障”为特征的软件构件模型。其中开放协同是基础，主要特征是“松耦合结构与分布式聚合”，它将突破传统封闭可控模型的限制，使得软件在结构上能够适应可重构环境对各类资源的多模式协同的要求。外界驱动的特征是“感知环境变化与动态重构”，其中外界环境变化也包括人为发布的重构指令，在此基础上，建立开放协同模型与环境模型的交互计算模式，从而形成环境驱动的动态化重构。更进一步，可信化保障主要指“数据的安全交互”，它是在环境驱动模型的基础上引入安全封装，解决开放环境下构件的可信性问题，最终形成一个安全可信的软件模型。

构件模型由构件和容器组成，构件分为构件代理和行为构件 2 种类型。容器负责外界环境的感知为构件提供运行环境，构件代理具有决策功能，构件模型中只含有一个构件代理，行为构件负责实现具体的功能。宏观上构件模型如图 1 所示。构件植入到容器后，首先向构件代理(CA, component agent)注册，由 CA 记录构件的属性信息并分配构件一个在容器内唯一的通信地址。构件间的交互是通过 AC(agent client)进行的，AC 利用容器提供的通信服务将构件的信息在构件间按需地进行传递。

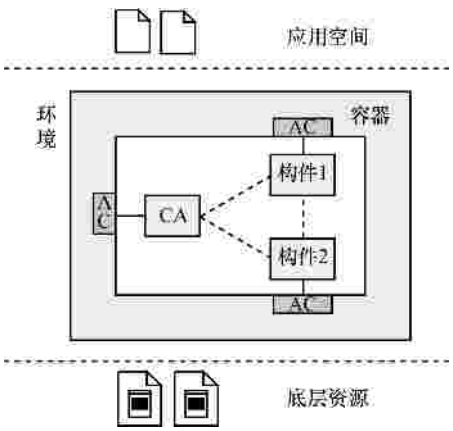


图 1 构件模型与外界环境关系

3.1　感知、决策和执行分离机制

为了适应可重构环境的环境多变和重构特性，本文提出了构件模型中感知、决策和执行分离的机制，如图 2 所示。感知功能包括对环境变化的检测和接收人为发出的重构指令，与外界环境交互密切所以交由容器实现，以建立环境模型和触发重构决策。本文将环境的变化抽象为软件运行的上下文变化，上下文是构件模型中一个重要概念，将在后文中详细阐述，它为形式化描述环境变化提供了方法和工具，便于容器感知功能的实现。容器除了具有感知功能，还负责完成构件对外信息的交互和对底层资源的调用，这点与软件中间件功能类似。

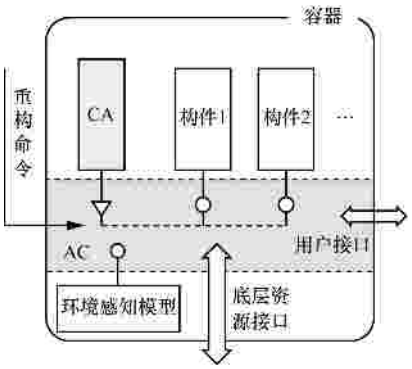


图 2　感知、决策和执行分离机制

决策功能是构件模型的中枢部分，是执行软件重构过程中决策逻辑的构件，因此本文使用一种特殊的构件实现决策功能，称为构件代理(CA)。本身这种实现方式也是一种功能分离机制，当构件模型决策机制变化时只需修改其中的构件代理，这样做可以使构件模型达到最大程度的前向兼容效果。构件代理的动作可以分为修改构件功能描述与修改构件连接拓扑：前者指构件代理可以对行为构件的语义进行调整，具体表现为构件属性描述信息的变化；后者则是指构件代理能够对行为构件间连接拓扑进行调整，如增删、替换构件和连接关系等，具体表现为构件代理修改容器所提供的 AC 服务。

行为构件用来在容器中实现软件的功能，大多数构件由第三方开发，需要考虑行为构件的安全性和标准化。因此后文重点关注行为构件接口的标准化和可信数据传递。行为构件是可重构操作的对象，本文正是通过行为构件的增删与替换实现软件整体功能的重构。

3.2　软件构件模型支持的重构机制

可重构柔性网络的特点是按照用户需求构建具有不同业务属性的可重构服务承载网，不同的服务承载网对应一系列不同的构件。服务承载网的构建和重构一方面涉及数据平面转发和交换部件的重构，另一方面涉及控制平面中软件构件的替换和增删。重构过程中两者相互配合，数据平面提供构件模型与外界交互的通道，控制平面借助构件模型完成软件构件的部署和替换，启用适应数据平面的控制构件。网络节点的重构过程涉及到节点内的构件代理、容器和行为构件，还需要节点外构件库的支持，执行过程如图 3 所示。服务承载网重构反映的是构件模型接收重构指令进行重构操作的过程，除此之外构件模型还可根据感知环境的变化进行构件重构，两者相比区别仅在于驱动重构的来源不

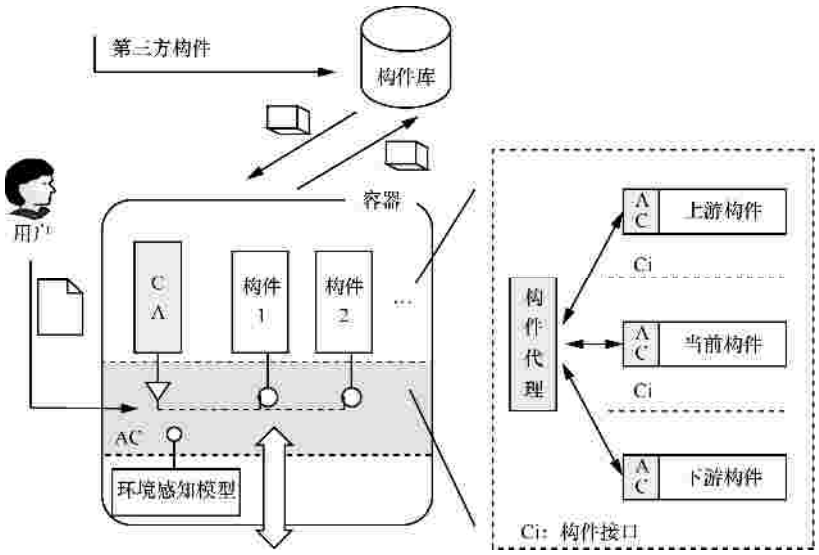


图 3　构件模型支持的重构机制

同，本文对前者进行详细描述，后者不再赘述。

结合前文的描述，用户负责下发重构方案，重构方案中包含了预先定义便于构件代理识别的重构指令。构件库中提前存有第三方遵循构件模型标准开发的构件，构件代理根据重构指令下载合适的构件。

构件运行时，首先向构件代理注册，构件代理为每个构件分配通信地址，并告知当前构件的通信对象。每个构件将与自己通信的构件分为上游构件和下游构件，接收对方消息时标记对方为上游构件，向对方发送消息时标记对方为下游构件。分别在各自 AC 中保持上、下游构件通信列表。当通信对象发生变化时只需更新 AC 中的通信列表。

当需要添加构件时，构件代理为新构件分配地址，同时广播新构件的地址，各构件 AC 根据广播消息更新自己的通信列表；删除构件时，构件代理将原有构件地址宣布为无效地址，各构件 AC 将自己通信列表中的相同地址标记为无效；替换构件时，只需将被替换构件 AC 中的信息迁移到新构件中。上述重构机制在实际场景中有多种实现方式，可根据编程语言选择合适通信机制和编程实现。

4 TCM 的构件类型和构造方法

构件模型是基于构件的软件开发方法的核心，它由构件语义、构件语法和构件组装 3 部分组成。构件语义说明了构件到底是什么；构件语法规定了构件是如何被定义、构造和表示的；构件组装则说明构件是如何被静态和动态组装的。

上下文是环境某一个维度(如可用内存、温度、位置等)，其定义可以来源于相应的上下文本体，上下文的值代表了环境在该维度的状态，上下文事件代表的则是该维度状态的变化。在上述定义的基础上，可以区分构件的上下文端口和服务接口；前者可以定义上下文事件，用来更新容器内部的环境模型等；后者则是一般意义上包括方法、属性等的接口。

定义 1 上下文 E 可定义为三元组 $<name, tpye, value>$，其中，$name$ 是上下文的名称，$type$ 是上下文的类型，上下文 E 在任一确定的时刻均有一个类型为 $type$ 的值 $value$。

定义 2 容器 C 可以定义为三元组 $<In, Out, service>$，其中，In 表示容器用来接收构件消息的端口集合，端口的类型决定了容器中可以植入构件的类型，Out 表示容器输出消息的端口集合，$service$ 表示容器可以提供的服务类型的集合，一般 $service$ 都会与一组 In 和 Out 端口绑定。

容器 C 是构件运行的环境，在软件生存周期内屏蔽环境变化造成的影响，它提供了丰富的供构件调用的服务接口，完成构件间基本的信息交互和对底层资源的调用，如内存分配和时钟管理等。容器 C 定义的端口和服务是第三方开发构件时参照的重要标准，容器 C 的实现方式决定了构件植入方式。

4.1 构件类型

构件代理是一类特殊构件，它不参与软件功能的处理，是对软件决策逻辑手段的封装，执行重构操作。构件代理负责接收用户制订的重构方案，按照预先约定的语义解析方案，根据方案进行构件重构，可以控制构件的加载，激活，配置和卸载。构件代理还可根据环境变化调整构件之间的连接关系。

对于行为构件引用传统意义的定义，即具有相对独立功能、可以明确辨识、接口由契约指定、可独立部署、和语义有明显依赖关系、且多由第三方提供的可组合软件实体[25]。它对具体业务处理行为进行封装，通过 AC 对外提供多个服务接口，并可以依赖其他接口。

4.2 构件构造方法

在已有的构件模型中，构件外部特征的定义方法可以分为两大类：直接使用具体的编程语言(如 Java)或使用专门的构件定义语言(如 CCM，COM/DCOM)。后者一方面有助于实现不同语言、不同平台间的互操作，更重要的是，它们可以显式地表示构件类型信息，从而有助于对软件体系结构的后续修改。因此，本文在 TCM 中引入了 XML 语言来定义构件。

本文对 XML 语言进行了扩展，主要的扩展内容包括：引入 component，id 等关键字及相关语法来定义构件；引入 event 关键字及相关语法来定义上下文事件；引入 in_port，in_port_num，in_message_type, out_port 等关键字来描述接口；引入 state 关键字来定义构件组装类型等。

通过专门设计的编译器可以将 XML 构件定义映射为 Java、C 等具体编程语言下的构件框架，构件开发者可以填写构件实现并生成构件分组。构件模型的接口中包括了普通服务、事件处理和构件映射 3 类方法，其中，普通服务和事件处理由 XML 定义直接映射生成，构件映射方法则需要向构件代理注册构件类型、当前运行状态等基本信息，由构件代理决定谁来具体完成该请求。TCM 构件的构造方法如图 4 所示。

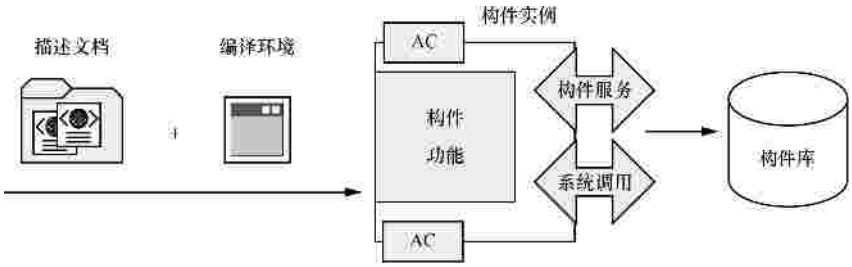


图 4　构件构造方法

4.3　构件组装机制

组装是构件模型应用的核心，构件模型可对行为构件按照满足应用需求的方案进行组装来实现软件功能。

在构件的组装过程中，涉及到不同构件之间的连接拓扑，而且不同构件的连接拓扑也决定了构件间信息流的传输路径。根据信息流在互连构件间的流向，从复杂的构件连接方式中提取出顺序组装、聚合组装和分支组装 3 种最基本的构件组装方式，如图 5 所示，图中省略了连接拓扑的实现细节。任何复杂的构件组装均以这 3 种组装关系为基础。从软件体系结构描述角度出发，可在基本构件的形式化描述基础上，推导出基于以上 3 种方式组装的复合构件的形式化描述。若以推导出的复合构件为基础构件，可进一步推导出更高层次的复合构件，如此反复，最终可推导出整个复杂软件系统的体系结构描述。TCM 模型从软件体系结构的角度实施构件的组装，通过 AC 的连接拓扑变化来实现构件间的组装，可以用 XML 语言描述构件间接口关系的改变来描述构件的组装蓝图。图 6 展示了使用 XML 语言描述简单的顺序组装和聚合组装。

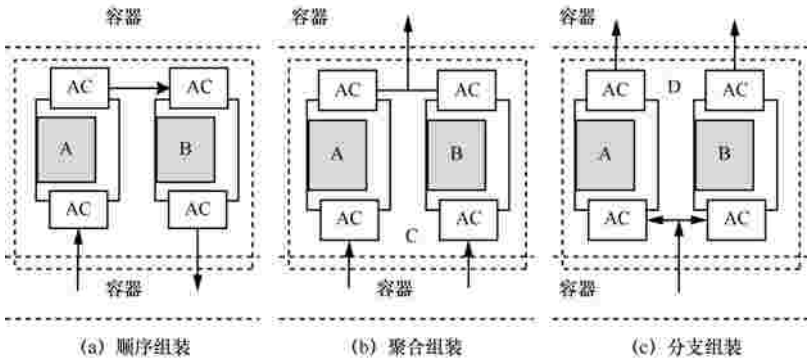


图 5　构件组装机制

```
<component>
    <name>A</name>
    <name>B</name>
    <id>A</id>
    <id>B</id>
<state>order assemble</state>      //组装类型

<event>
    <AC>A.AC->B.AC</AC>
</event>                           //描述组装关系

<in_port>                          //组装后的输入端口
    <in_port_num> A.port</in_port_num>
    <in_message_type>A_message</ in_message_type>
</in_port>

<out_port>                         //组装后的输出端口
    <out_port_num> B.port</out_port_num>
    <out_message_type>B_message</out_message_type>
</out_port>
```

(a) 顺序组装描述

```
<component>
    <name>C</name>
    <name>A</name>
    <name>B</name>
    <id>A</id>
    <id>B</id>
    <id>C</id>
<state>conglomeration assemble</state>      //组装类型

<event>
    <AC>C.AC=A.AC+B.AC</AC>
</event>                           //描述组装关系

<in_port>                          //组装后的输入端口
    <in_port_num> C.port=A.port</in_port_num>
    <in_port_num> C.port=B.port</in_port_num>
    <in_message_type>C_message=A_message</ in_message_type>
    <in_message_type>C_message=A_message</ in_message_type>
</in_port>

<out_port>                         //组装后的输出端口
    <out_port_num> C.port</out_port_num>
    <out_message_type>C_message</out_message_type>
</out_port>
```

(b) 聚合组装描述

图 6　XML 语言描述的构件组装

5 TCM 模型的可信机制

可重构柔性网络节点中运行着大量第三方构件，其中，多个构件在容器中组装成为更复杂的软件构件。构件模型的可信机制应既要保证单个构件的可信，也要保证组装后构件系统的可信性。

构件在构件模型中通过 AC 调用容器提供的接口进行消息通信，在 TCM 模型中提出了一种构件接口数据的安全封装机制，防止构件进行违反约束的访问和操作。容器是将构件和底层操作系统隔离的一种构件运行环境，操作系统默认容器与自己进行的操作都是高安全级别的可信操作，构件进行的操作都是低安全级别的操作。低安全级别的操作可以依赖高安全级别的操作，若高安全级别的操作依赖于低安全级别的操作则认为这是不可信的。可信机制中涉及的对象包含CA代表的构件代理, C代表的容器，AC 代表的构件接口。本文在容器中对构件的接口数据进行安全属性划分和封装。

定义 3 安全级别、安全操作和安全操作集函数。利用安全级别对 CA、C、AC 进行划分。CA 和 C 之间交互的数据属于高安全级别，记为 $data_{\mathrm{H}}$，AC 之间交互的数据属于低安全级别，记为 $data_{\mathrm{L}}$。CA 和 C 进行的操作属于高安全级别操作，记为 Act_{H}，AC 进行的操作属于低安全级别操作，记为 Act_{L}。函数 $get\_Acts(d:data)\rightarrow 2^{Acts}$ 表示由数据安全级别得到执行该数据安全操作的集合。

定义 4 安全属性和安全属性映射函数。安全属性记为 $prop$，安全映射函数记为 $fprop(p:prop)\rightarrow data$。

在容器 C 中，将存储空间划分为相互隔离的区域使用安全属性标记，为了保证构件传递数据的安全，使用安全映射函数将构件的数据保存在不同的存储空间。当发生构件组装后，将组装在一起的构件数据映射到安全属性相同的存储空间，如图 7 所示。

构件代理为相互通信的构件分配相同 security property 值，AC 将构件的数据与此 security property 值绑定。容器根据构件数据的 security property 值将数据存储在特定的安全存储空间。当其他构件读取此数据时，只有 security property 值一致时才能获取数据。构件组装后，构件代理为被组装的构件分配同一个 security property 值，构件只能使用此 security property 与参与组装的构件通信。这种机制可以保证构件代理对构件间通信的管理，避免恶意构件自主通信。security property 值由构件代理分配，取值来自 $data_{\mathrm{H}}$ 和 $data_{\mathrm{L}}$ 构件无权修改，AC 负责绑定 security property 值和数据，恶意构件无法伪造 security property 值欺骗 AC。另外还可以使用加密算法保证 security property 值不被恶意构件获取。

6 原型验证及实验

本节描述在可重构路由器中实现的 TCM 容器原型，并基于容器开发的构件实例来验证 TCM 模型的有效性。

6.1 支持 ATCM 的容器原型

容器除了提供构件运行的环境外，也是实现重构的场所和提供可信机制的基础设施。基于 Quagga 路由软件的 Zebra 平台[26]实现了一个 TCM 的容器原型，其主要部件如图 8 所示。构件管理引擎负载构件的加载、激活和卸载等操作。通信引擎负责支持构件的数据通信和转发构件的服务接口的访问请求，提供数据报文的传递。可信存储提供数据安全存储机制，隔离构件低安全级别的操作。构件组装引擎负责执行和维护构件的组装，并将组装结果反馈给可信存储。环境模型提供外界环境变化的表征，保存当前影响构件的环境维度的属性值，构件属性描述信息等。日志记录负责保存重构操作记录，容器架构信息等。系统接口与操作系统交互，

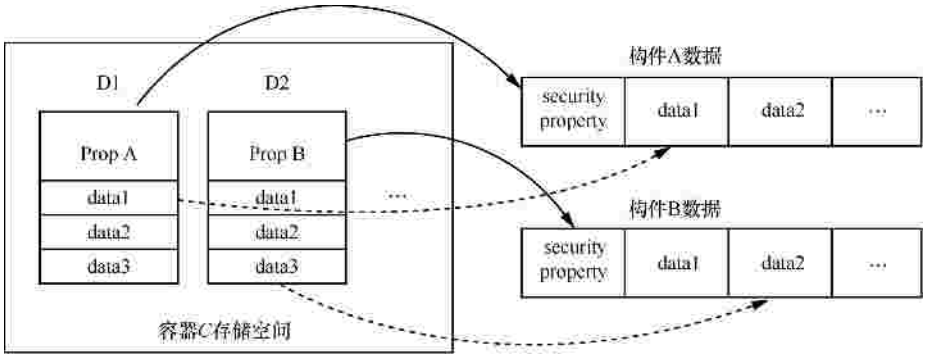


图 7 容器 C 安全存储资源分布

为容器提供操作系统的支撑服务。

Zebra 容器运行时需要维护系统内部状态的一致性，如构件重构时的状态迁移等。当构件模型发生变化时，构件管理引擎负责映射构件相关的变化，环境模型负责映射容器体系结构的变化。当发生如构件意外失效的特殊情况时，构件管理引擎通过轮询构件 AC 反射接口获知情况变化，相应的状态会被反馈到环境模型中。Zebra 容器目前可以运行在 Linux 和其他 Unix 变体系统上，符合 GNU 的 GPL 标准并提供了支持 C 语言编译器等开发工具。

除了上述构件重构和管理部分的功能外，Zebra 容器本身对路由软件提供了非常丰富的操作支持。它支持 BGP-4、BGP-4+、OSPFv2、OSPFv3、RIPv1、RIPv2 和 RIPng 等协议的标准接口和数据操作，对外提供类似 Cisco IOS 管理操作接口。第三方开发者可以按照构件 AC 和容器接口规范开发路由协议构件，在 6.2 节将介绍构件 AC 接口的使用。

6.2　构件模型重构实例

以 OSPFv2 协议构件和路由管理构件为例介绍构件的重构过程，图 9 中是 OSPFv2 协议的一种实

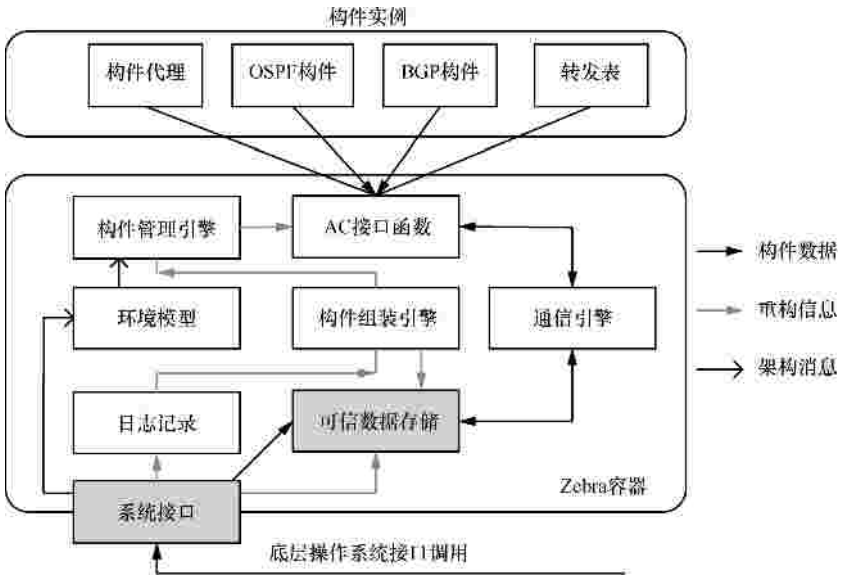


图 8　ATCM 容器原型

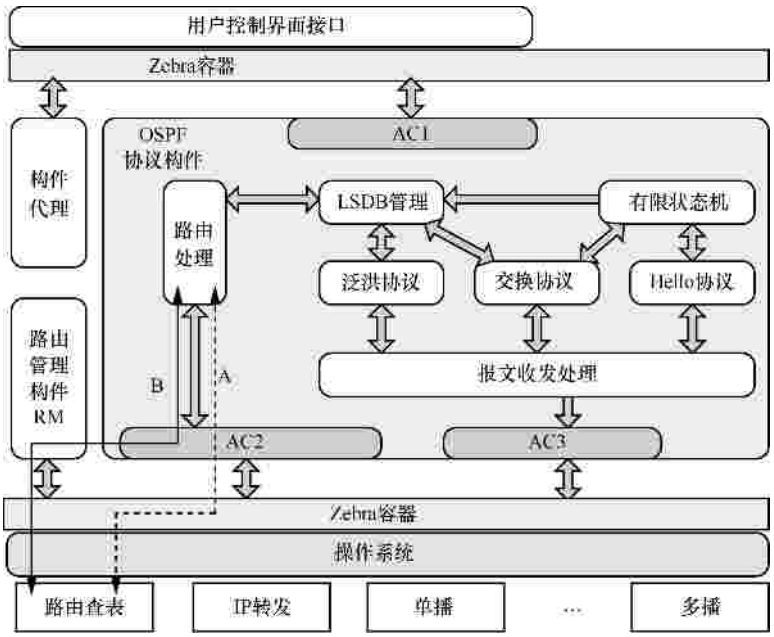


图 9　路由协议构件运行示意

现实例。第三方开发 OSPFv2 构件时首先满足协议处理功能需求，其次根据构件模型规范，开发与构件模型需要交互的接口，开发时使用 3 类 AC，AC1 负责为网络管理员提供一个通过指令配置、干预、查看 OSPFv2 运转状况的通道；AC2 提供函数可获取底层接口信息和重分配路由信息，同时将 OSPF 路由信息交给路由管理构件；AC3 主要实现 OSPFv2 协议构件与操作系统的 TCP/IP 协议栈的连接，以接收和发送数据分组，并提供对协议模块运行过程中的内存和定时器管理。

通过如下场景验证构件模型对构件重构的支持。

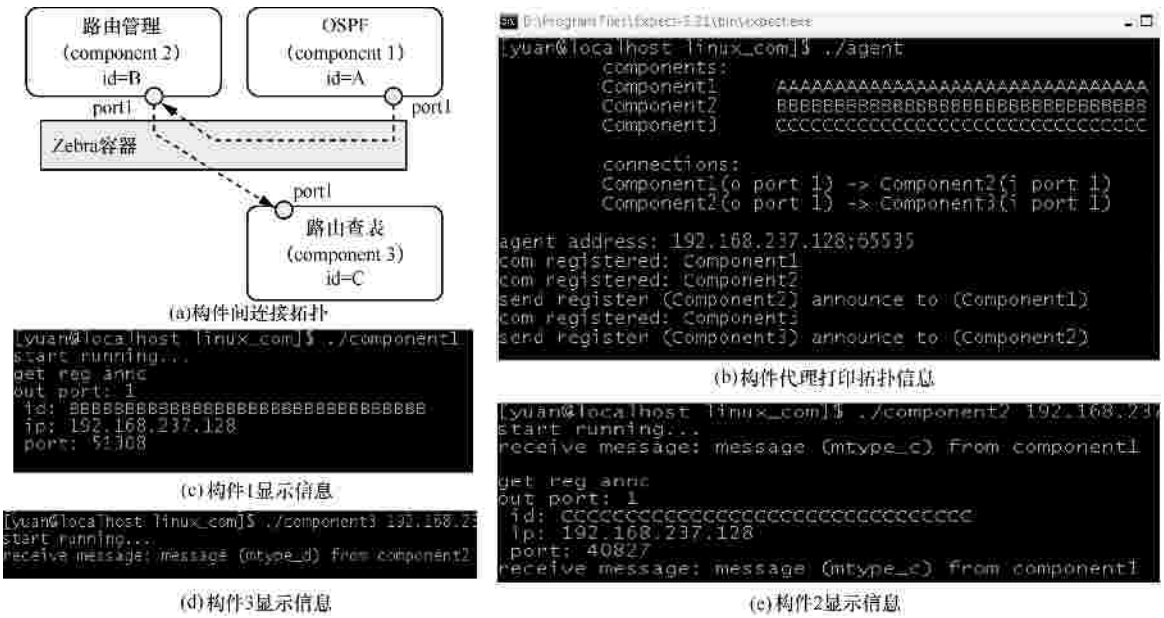
1) 构件的加载。初始情况下路由设备中只启动 Zebra 容器。构件代理发布命令下载和启动 OSPFv2 构件(component 1)，OSPFv2 构件可以进行正常的协议报文收发，并且将协议处理的路由信息下发给底层路由查表模块。如图 9 路径 A 所示。图 10 显示了构件的端口信息和构件代理打印的构件拓扑连接信息。其中，路由查表构件(component 3)属于硬件构件，虽然不在 Zebra 容器中运行但可以接收构件代理的消息。

2) 构件间连接关系的重构。随着路由设备的运行，陆续加载 RIP，BGP 等构件，此时需要路由管理构件来统一管理各个路由协议的路由信息，此时需要改变 OSPFv2 构件路由信息的传输路径，如图 9 路径 B 所示。构件代理下发重构命令，修改构件的通信关系同时更改构件的 security property 值。图 11 显示了构件代理打印的构件注册信息和构件拓扑连接信息，以及各个构件间的消息传递。此时构件都被设置使用端口 1 传递消息。

3) 构件的卸载。构件运行中如果需要对构件进行替换，需要将旧的构件卸载，通过构件代理发布命令卸载路由管理构件(component2)。构件显示的信息如图 12 所示。可以发现，构件代理发布命令，注销 component 2，同时将此消息通知 component 1

(a)构件连接拓扑　(b)构件打印拓扑信息

图 10　构件间拓扑连接

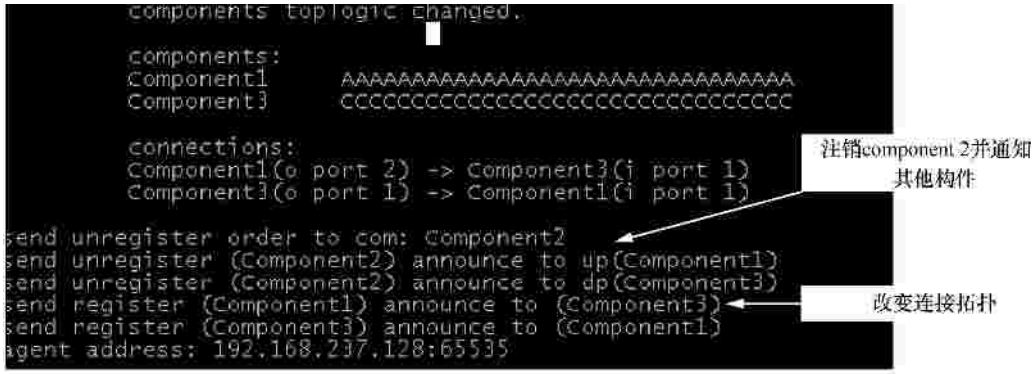


(a)构件间连接拓扑　(b)构件代理打印拓扑信息

(c)构件1显示信息　(e)构件2显示信息

(d)构件3显示信息

图 11　重构构件时消息传递显示



(a) 构件代理打印信息

(b) component 2停止运行

(c) component 3接收到component 1的消息

图 12　卸载构件过程中的信息显示

和 component 3，随后通知 component 1 和 component 3 改变通信对象，从而实现了构件连接拓扑关系的改变。

6.3　构件模型可信机制评估

构件模型在 Zebra 容器中内嵌安全存储机制来实现构件交互数据的安全可信。设计 2 个场景评估可信机制的效果，一个是安全存储对可重构路由器的构件性能的影响，即对单位时间内报文交互时延的影响。另一个场景是测试可信机制对恶意行为检测效果。

场景 1　使用路由器 R1 将真实路由分别广播给启用安全存储的可重构路由器 R2 和不启用安全存储的可重构路由器 R3，待 R2 和 R3 学得路由后再分别广播给路由器 R4。在 R1 和 R4 的链路中抓取路由广播分组，抓取分组软件使用 Etheral 0.10.14，计算分组时延并进行比较，观察可信存储机制造成的时延。普通路由器 R1 和 R4 使用的是中兴 ZXR10，路由表数据来自 routing information service(RIS)[27]。测试网络拓扑如图 13 所示。

路由器处理每条路由的时间较短，选择真实路由表中前缀为 25 的路由信息 10 000 条，每 100 条统计一次均值，并且设置 OSPFv2 构件每 5s 广播一次路由信息，分别用 c、a 点的时间戳差值和 d、b 点的时间戳差值比较。

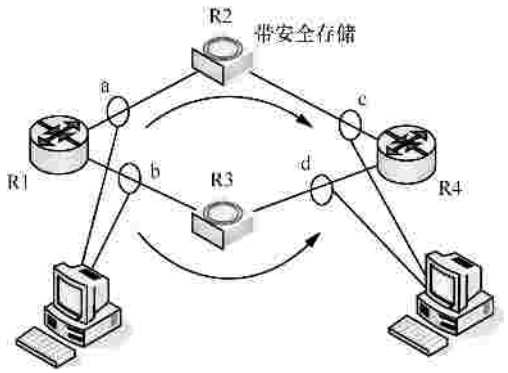


图 13　测试网络拓扑

可重构路由器中使用最多的是路由协议构件，OSPFv2 构件可以设置广播路由消息的时间间隔，固定的时间间隔削弱了路由信息的实时处理性，为消息的安全封装和读写提供了时间保证。从图 14 中可以发现虽然路由信息的处理时延存在抖动但是控制在 1s 之内，2 个路由器的处理时延平均值相差不到 0.4s，这个时间差主要产生于从安全数据存

储区域取出数据广播路由信息这个过程。构件模型的数据安全存储机制对路由信息报文交互的处理时延影响较小，不会影响路由协议的性能。

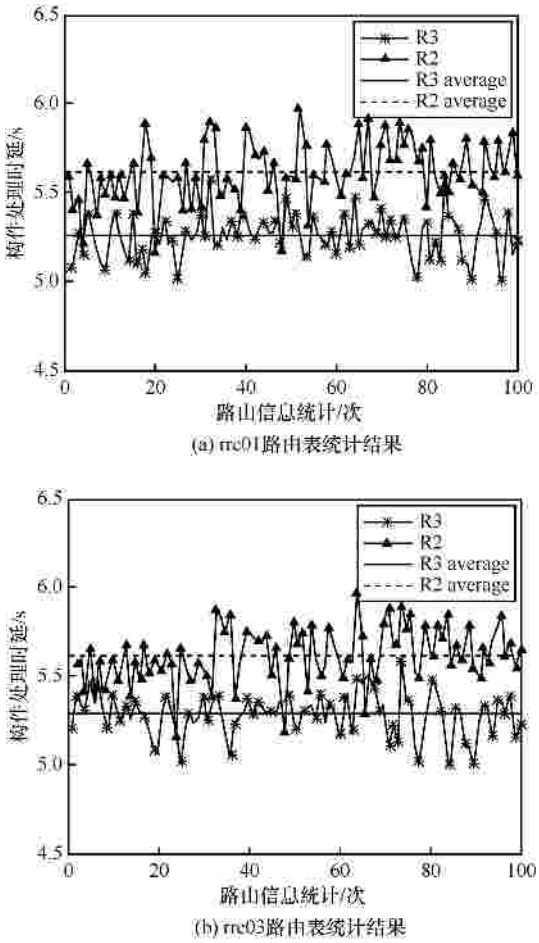


图 14 R2 和 R3 路由信息处理时延比较

场景 2 构件恶意行为检测。

使用 rip 构件充当恶意构件，修改 rip 构件的协议报文发送函数，使它冒充 OSPFv2 构件调用 AC 提供的接口函数进行报文收发。下面的这些回调函数的赋值用来完成构件代理对 OSPFv2 构件的控制，原来包含在 OSPFv2 构件中。OSPF 协议中使用的是 raw socket，所以修改时重点替换的是 socket 创建和 setsockopt 函数(如图 15 所示)。篇幅所限不在此一一列举。

Zebra 容器运行过程中，原本发送给 OSPFv2 构件的数据被恶意的 rip 构件调用 AC 接口函数进行读写，虽然此时 AC 接口函数可以被调用，但是当恶意构件在安全存储区读取数据时，无法获得正确的 security property 值，读取操作被拒绝，并在反馈给构件代理警告信息，运行结果如图 16 所示。

```
Zebra_Component_Admin_Interface ospf_admin_intf = {
    Ospf_callback_init,
    Ospf_callback_stop,
    Ospf_callback_poll,
    Ospf_callback_read,
    Ospf_callback_write,
    Ospf_callback_showrun,
    Ospf_callback_statics,
    Ospf_callback_vendor,
};
```

(a) 回调函数修改

```
在rip 构件中注释掉
    //rip_if_init ();
    //rip_zebra_init ();
添加
    ospf_if_init ();
    ospf_zebra_init ();
```

(b) AC接口函数修改

图 15 构件代码修改示例

图 16 构件恶意行为检测

从结果中可以发现，构件注册时构件代理把恶意 rip 构件认为是 OSPFv2 构件，所以恶意构件注册成功并运行，但是当构件开始读取数据时被发现为恶意构件，结果被存入日志并且上报构件代理，构件代理将其打印显示，同时暂停恶意构件的运行待管理人员进行处理。其他构件不受此影响正常运行。

7 结束语

构件化的路由设备是可重构柔性网络的重要节点支撑设备。本文以路由设备控制平面支持可重构为背景，兼顾软件系统对环境变化的适应。基于构件间连接关系的调整和构件数据的安全存储提出了一种可信构件模型 TCM。构件模型从环境感知、决策、执行三者相分离的思想出发，使用构件代理完成决策功能，使用容器感知环境和隔离底层操作系统影响，使用行为构件实现软件系统功能，通过在容器中内嵌安全存储机制提高构件的可信

性。根据本文构件模型开发的 Zebra 容器和在其中实现的构件实例已经应用到了可重构路由器的开发中。后续工作将对深层次的构件重构机制和构件的可信评估展开研究。

参考文献：

[1] **汪斌强，邬江兴. 下一代互联网的发展趋势及相应对策分析**[J].**信息工程大学学报**, 2009, 10(3):1-6.
WANG B Q, WU J X. Development trends and associated countermeasures analysis for NGN[J]. Journal of Information Engineering University, 2009, 10(3):1-6.

[2] **袁博，汪斌强，张博. 绿色网络的实例——可重构柔性网络**[J]. **电信科学**, 2011, 27(10A):200-207.
YUAN B, WANG B Q, ZHANG B, A case study of green network ——reconfigurable flexible network[J]. Telecommunications Science, 2011, 27(10A):200-207.

[3] MEI H, SHEN J R. Progress of research on software architecture[J]. Journal of Software, 2006, 17(6):1257-1275.

[4] VALLECILLO A, HERNANDEZ J, TROYA J M. Component Interoperability[R]. ITI200037, Universidad de Málaga, 2000.

[5] YANG F Q. Thinking on the development of software engineering technology[J]. Journal of Software, 2005,16(1):1-7.

[6] SALEHIE M, TAHVILDARI L. Self-Adaptive software: landscape and research challenges[J]. ACM Trans on Autonomous and Adaptive Systems, 2009,4(2):1-42.

[7] WANG Q X. Towards a rule model for self-adaptive software[J]. Sigsoft Software Engineering Notes, 2005, 30(1):1-5.

[8] KEENEY J. Completely Unanticipated Dynamic Adaptation of Software[D]. Dublin: Trinity College, University of Dublin, 2004.

[9] DOWLING J, CAHILL V. The K-component architecture meta-model for self-adaptive software[A]. Proc of the Int'l Conf on Meta level Architectures and Separation of Crosscutting Concerns[C]. Kyoto, Japan, 2001. 81-88.

[10] PASPALLIS N, PAPADOPOULOS G A. An approach for developing adaptive, mobile applications with separation of concerns[A]. Proc of the 30th Annual Int'l Computer Software and Applications Conf (COMPSAC)[C]. Chicago, USA, 2006. 299-306.

[11] OREIZY P, GORLICK M M, TAYLOR R N, *et al*. An architecture-based approach to self-adaptive software[J]. IEEE Intelligent Systems, 1999, 14(3):54-62.

[12] GARLAN D, CHENG S W, HUANG A C, *et al*. Rainbow: architecture-based self-adaptation with reusable infrastructure[J]. IEEE Computer, 2004, 37(10):46-54.

[13] MOREIRA R, BLAIR G, CARRAPATOSO E. FORMAware: framework of reflective components for managing architecture adaptation[A]. Proc of the 3rd Int'l Workshop on Software Engineering and Middleware[C]. Orlando, USA, 2002.

[14] BRUNETON E, COUPAYE T, STEFANI J B. Recursive and dynamic software composition with sharing[A]. Proc of the 7th Int'l Workshop on Component-Oriented Programming (WCOP)[C]. Malaga, Spain, 2002.

[15] YANG F Q, LV J, MEI H. Technical framework for Internetware: an architecture centric approach[J]. Science in China (Series E), 2008, 38(6):818-828.

[16] MEI H, HUANG G, ZHAO H Y, *et al*. A software architecture centric engineering approach for Internetware[J]. Science in China (Series E), 2006, 36(10):1100-1126.

[17] LU J, MA X X, TAO X P, *et al*. On environment-driven software model for internetware[J]. Science in China (Series E), 2008, 38(6): 864-900.

[18] VMware, Inc VMware virtual machine technology[EB/OL]. http://www.vmware.com, 2006.

[19] GARFINKEL T, PFAFF B, CHOW J, *et al*. Terra: a virtual machine-based platform for trusted computing[A]. Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03)[C]. New York, USA, 2003. 193-206.

[20] KARGER P A. Multi-level security requirements for hypervisors[A]. Proceedings of the 21st Annual Computer Security Applications Conference (ACSAC'05)[C]. Tucson, Arizona, USA, 2005. 267-275.

[21] JAEGER T, SAILER R, SREENIVASAN Y. Managing the risk of covert information flows in virtual machine systems[A]. Proceedings of the 12th ACM Symposium on Access Control Models and Technologies (SACMAT'07)[C]. Sophia Antipolis, France, 2007. 81-90.

[22] CABUK S, DALTON C I, RAMASAMY H G, *et al*. Towards automated provisioning of secure virtualized networks[A]. Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS'07)[C]. Alexandria, VA, USA, 2007. 235-245.

[23] STEFAN B, RAMON C, KENNETH A G, *et al*. vTPM: virtualizing the trusted platform module[A]. Proceedings of 15th USENIX Security Symposium[C]. Vancouver, Canada, 2006. 305-320.

[24] MELVIN J A, MICHA M, CHRIS I D. Towards trustworthy virtualization environments: Xen library OS security service i frastructure[EB/OL]. http://www .hpl.hp.com/techreports/2007/HPL- 2007-69.html, 2007.

[25] **杨芙清，梅宏，李克勤. 软件复用与软件构件技术**[J]. **电子学报，**1999, 27(2):68-75.
YANG F Q, MEI H, LI K Q. Software reuse and software component technology[J]. Acta Electronica Sinica, 1999, 27(2):68-75.

[26] QUAGGA guide[DB/OL]. http://quagga.net.

[27] RIS raw data[DB/OL]. http://data.ris.ripe.net.

作者简介：

袁博（1981-），男，山东济南人，国家数字交换系统工程技术研究中心博士生，主要研究方向为可重构网络、构件可重构技术。

汪斌强（1963-），男，安徽安庆人，博士，国家数字交换系统工程技术研究中心教授、博士生导师，主要研究方向为可重构柔性网络。

马东超（1980-），男，北京人，博士，北方工业大学讲师，主要研究方向为下一代互联网、下一代接入网、网络协议和应用。