基于π演算的软件体系结构一致性分析研究

任洪敏 刘 晋
(上海海事大学计算机系 上海 200135)

摘 要 体系结构是系统的高层抽象和集成蓝图,基于形式化方法描述和分析体系结构能在系统开发早期发现体系结构层面的错误并提升体系结构的质量。基于体系结构的π ADL形式化规约,结合体系结构领域的需求和特点,运用π演算基本理论形式化定义了系列概念和多种进程关系,并以此作为理论基础,提出了π ADL规约的8种一致性分析方法,用以提高体系结构规约和系统组装的质量。

关键词 软件体系结构,体系结构分析,形式化方法,π演算
中图法分类号 TP311 **文献标识码** A

Research on Software Architecture Consistency Analysis Based on π Calculus

REN Hong-min LIU Jin
(Department of Computer Science and Technology, Shanghai Maritime University, Shanghai 200135, China)

Abstract As the blueprint for software development, software architecture deals with design and implementation of high level structure of software systems. Describing and analyzing software architecture based on formal approach can substantially detect architectural mismatches in the early stage of system development and then improve the architecture quality effectively. Based on formal architectural description language π ADL and taken into consideration of the characteristics and requirements of software architecture, a collection of concepts and process relationships were defined formally using π calculus theory as a theory foundation, then eight static check methods to determine consistency of π ADL architecture specifications were developed, which can significantly raise the qualities of architectural specifications and system composition.

Keywords Software architecture, Architecture analysis, Formal method, π calculus

1 引言

软件体系结构描述系统的总体结构,是决定软件系统质量的重要因素。软件体系结构的评估和分析可以显著减少及避免软件错误的产生和后期纠错的高昂代价。在软件体系结构设计一级,能够执行多种系统属性分析,如系统功能分析、扩充性分析、临界资源竞争情况分析、系统实现代价分析[1]。然而,无论执行何种分析,体系结构的设计都必须满足一个基本的要求:一致性(Consistency)。体系结构的一致性是指其各个部分必须相互一致,而不能彼此冲突[2]。对体系结构而言,一致性尤其重要,因为体系结构主要关注系统的结构和组装,如果参与组装的各个部分之间彼此存在冲突,那么由此组装、精化和实现的系统一定不能正确工作。

体系结构分析的内容包括结构分析、功能分析和非功能分析,分析的方法包括定量分析和定性推断等。如AT&T贝尔实验室开发了检查表(Checklist)方法进行体系结构评审,用于发现体系结构中主要存在的问题[3]。Kazman[4]等人提出了SAAM体系结构分析方法,它基于场景技术,分析软件的非功能质量属性。而Barbacci[5]等人则提出了多质量属性情况下的体系结构质量模型、分析和权衡的方法ATAM,该方法的主要过程是生成质量属性树,找出体系结构中的敏感点(Sensitivity Point)和权衡点(Tradeoff Point),进行相关分析和得出结论。其它相关研究工作包括Debra等人提出的体系结构测试覆盖准则[6]、文献[7]给出的基于霍尔公理的构件设计正确性验证技术等。上述研究工作主要提供软件评估的方法、指导原则和评估过程等,它们通常独立于特定的体系结构规约方法。基于体系结构描述语言展开的体系结构分析方法的研究工作包括C2[8],Unicon[9]等能够进行类型一致性和拓扑结构约束的检测。Darwin[10]基于它的π语义模型分析实现系统配置的分布式算法。Rapide[11]运用事件监控和过滤工具进行系统行为仿真,从而能够对系统相关行为进行分析。Wright[2]基于CSP描述提供了丰富的体系结构静态分析方法,它能保障体系结构规约的质量,这也是Wright的特点所在。

本文基于软件体系结构的形式化规约语言πADL[12],结合体系结构领域的需求和特点,运用π演算基本理论形式化地定义了系列概念和多种进程关系,并以此作为理论基础,全面开展软件体系结构规约一致性的形式化分析研究,具体包括体系结构3个基本要素即构件、连接件、配置的一致性分析,以及构件的替换性(Substitutability)分析和组装的兼容性

到稿日期:2010-05-18 返修日期:2010-12-07 本文受上海市教委创新基金(09YZ224),上海市自然科学基金(10ZR1413700)资助。
任洪敏(1969-),男,博士,副教授,主要研究方向为软件体系结构、基于构件的软件工程、形式化方法,E-mail:renhongmin@fudan.edu.cn。

(Compatibility)分析。

2 π ADL 体系结构描述语言简介

π ADL 基于多元 π 演算[13](Polyadic π-Calculus)描述软件架构，遵循 Wright[2]描述软件架构的框架，独立描述构件、连接件和系统配置。π ADL 运用 π 演算的进程描述构件端口、连接件角色和它们内部的计算行为。π ADL 在 π 演算的语法元素和架构行为描述所关注的内容之间建立映射，提供构件、连接件的控制行为、观察行为和内部行为与 π 演算语法元素的对应关系，从而给 π 演算语法元素赋予了架构级的语义。

π ADL 的具体语法规定和建模方法参见文献[13]，下面通过对客户-服务器架构风格中客户构件的描述，简要说明 π ADL 描述架构元素行为的方法。其中 Π 为 π ADL 引入的简便标记，表示非确定选择，其语义为 $P\Pi Q=\tau.P+\tau.Q$。

Component Client

Port Cp=$\overline{\text{request}}$. reply. Cp$\Pi$0

Computation=τ_ internalCompute. $\overline{\text{Cp\_request}}$. CP_ reply. Computation$\Pi$0

客户构件 Client 具有一个端口 Cp，它的行为交互协议是：发出服务请求$\overline{\text{request}}$，收到回答 reply 后，内部非确定地选择继续执行进程 Cp 或决定结束服务请求而执行进程 0。Client 的计算行为是：首先进行内部计算 τ_internalCompute，然后通过端口 Cp 发出服务请求$\overline{\text{Cp_request}}$，等到端口 Cp 得到回答 Cp_reply 后，Computation 继续进行该过程或决定结束。

运用端口名称修饰、限定通道名称，防止各个端口的通道同名，因此$\overline{\text{Cp_request}}$和 Cp_reply 表示端口 Cp 发出服务请求$\overline{\text{request}}$和得到回答 reply。

3 基本概念

π 演算的理论基础是通过互模拟机制定义的行为等价理论。如果两个进程行为等价，则在任意环境中，用一个进程替换另一个进程都不能从外部发现产生了差别，从而不能以此判定是否发生了替换。在体系结构研究领域，行为等价理论过于严格。因为在进行构件替换、端口角色连接时，通常两者的行为并不完全等价，而是可能不同。因而 π 演算的行为等价理论并不完全适用于体系结构分析。基于 π 演算的基本理论，结合软件体系结构领域分析的需求和特点，首先给出 πADL 进行体系结构分析需用的概念定义和理论基础。

定义 1(基本定义)　设名字的集合是 $\mathbb{N}$。进程中，出现在输入前缀和限定操作符中的名字称为进程的限定名字，即 $x(\vec{z}).P$ 和 $\overline{vz}.P$ 中，在 $\vec{z}$ 中的名字为限定名字，其它的名字称为自由名字。进程 P 中自由出现名字的集合记为 $fn(P)$。

定义 2(进程变换)　设函数 $f:\mathbb{N}\to\mathbb{N}$ 和进程 P，则 $\hat{f}(P)$ 表示如下进程：把进程 P 中所有自由出现的名字 α 替换成为 $f(\alpha)$、$\bar{\alpha}$ 替换成为 $\overline{f(\alpha)}$ 而得到的进程。f 称为符号变换函数。

定义 3(名字限定变换函数)　设任一进程 P，如下符号变换函数：

$$PT_x(\alpha)=\begin{cases}x\_e, & \text{if } \alpha=e\wedge e\in fn(P)\\ \overline{x\_e}, & \text{if } \alpha=\bar{e}\wedge\bar{e}\in fn(P)\\ \tau\_x\_e, & \text{if } \alpha=\tau\_e\\ e, & \text{otherwise}\end{cases}$$

称为名字限定变换函数。$\hat{PT}_x(P)$ 表示运用符号 x 作为进程 P 中事件的前缀而生成的进程。

定义 4(名字隐藏)　给定进程 P 和名字集合 E，$E\subseteq fn(P)$，P/E 表示隐藏、屏蔽进程 P 中所有在集合 E 中的名字后而得到的进程。设 E 中的名字为 $e_1,e_2,\cdots,e_n$，则名字隐藏形式化定义如下：

$$P/E=vE(P|Ever(e_1)|Ever(e_2)|\cdots|Ever(e_n))$$

$$Ever(x)=x(y).(Ever(x)|Ever(y))+vy(\bar{x}y).(Ever(x)|Ever(y))$$

名字隐藏是把一个进程中的相关名字进行隐藏，做法是把进程中的该名字运用内部行为 τ 进行替换。$Ever(x)$ 的直观含义是提供名字 x 的输入前缀或输出前缀，它与 P 组装后，与 P 中的 x 的输出前缀或输入前缀进行交互，约简成为内部行为 τ，从而达到隐藏 x 的目的。

定义 5(进程死锁)　设 π 演算进程 P，存在进程 P'，并且如下条件成立：

$$P\Rightarrow P'\wedge\{P'\to P''\}=\emptyset\wedge P'\neq 0$$

则进程 P 存在死锁。进程 P 存在死锁，表明进程 P 能够内部演化到进程 P'，然后再也不能执行内部演化，然而其行为并未完全执行，即 P' 并不结构等同于进程 0。

实践中系统组装时，两个构件的端口通常直接相连，为了判断两个构件的端口能否直接相连，定义进程兼容关系。

定义 6(进程半兼容)　设进程集合之上的二元关系 C，如果 pCq，则如下条件成立：

(1)如果 $p\xrightarrow{\tau}p'$，则 $p'Cq$。

(2)如果$\neg(P\Rightarrow 0)$，则：

(a)如果 $p\xrightarrow{x(w)}p'\wedge q\xrightarrow{\bar{x}y}q'$，则 $p'(y/w)Cq'$。

(b)如果 $p\xrightarrow{x(w)}p'\wedge q\xrightarrow{\bar{x}(w)}q'$，则 $p'Cq'$。

则称 C 为半兼容关系。如果 pCq，则称进程 p,q 半兼容。

定义 7(进程兼容)　如果 C 和 C^{-1} 都是半兼容关系，则称 C 为兼容关系，如果 pCq，则称进程 p,q 兼容，记为 $p\boxminus q$。进程 P,Q 兼容，意味着 $P|Q$ 不存在死锁。

定义 8(进程半替换)　设进程集合之上的二元关系 S，如果 pSq，则如下条件成立：

(1)如果 $p\overset{x(y)}{\Rightarrow}p'$，则 $\exists q':q\overset{x(y)}{\Rightarrow}q'\wedge p'Sq'$。

(2) 如果 $p\Rightarrow p'$，则 $\exists q':q\Rightarrow q'\wedge p'Sq'$。

(3)如果 $p\overset{\bar{x}_1y_1}{\Rightarrow}p_1,p\overset{\bar{x}_2y_2}{\Rightarrow}p_2,\cdots,p\overset{\bar{x}_ny_n}{\Rightarrow}p_n$，则 $\exists q',\bar{x}_iy_i(1\leqslant i\leqslant n):q\overset{\bar{x}_iy_i}{\Rightarrow}q'\wedge p_iSq'$。

(4) 如果$\neg\exists x,y,p':p\overset{\bar{x}y}{\Rightarrow}p'$，则$\neg\exists q':q\overset{\bar{x}y}{\Rightarrow}q'$。

(5)如果 $p\overset{\bar{x}_1(y_1)}{\Rightarrow}p_1,p\overset{\bar{x}_2(y_2)}{\Rightarrow}p_2,\cdots,p\overset{\bar{x}_n(y_n)}{\Rightarrow}p_n$，则 $\exists q',\bar{x}_i(y_i)(1\leqslant i\leqslant n):q\overset{\bar{x}_i(y_i)}{\Rightarrow}q'\wedge p_iSq'$。

(6) 如果$\neg\exists x,y,p':p\overset{\bar{x}(y)}{\Rightarrow}p'$，则$\neg\exists q':q\overset{\bar{x}(y)}{\Rightarrow}q'$。

则称二元关系 S 为半替换关系，如果 pSq，则称 q 可半替换 p，记为 $p\prec q$。

πADL 运用输出名字表达构件的启动行为，运用输入通道表达构件的观察行为。因此，条件(1)表示：如果进程 p 能够观察并响应行为 $x(y)$，则构件 q 同样能够观察并响应行为 $x(y)$，但在同时，并不限制 q 还可以观察和响应其它的行为。

条件(2)表示如果 p 能够执行系列内部行为演化到 p'，则 q 同样能够执行 0 个或多个内部行为演化到 q'，并且 p'，q' 同样满足关系 S，这是一个明显的条件，因为我们仅仅关心构件外部的行为，而不关心它的内部行为。而条件(3)表示：如果 p 能够选择执行多个启动行为，则 q 至少能够选择地执行其中一个或多个启动行为。条件(4)表示：如果 p 不能够启动执行行为 $\bar{x}y$，则 q 同样不能启动行为 q。条件(5)、(6)表达限定输出行为方面的条件，意义基本同于条件(3)、(4)。因此，$p<q$ 意味着：q 能够处理 p 的所有观察行为，但 p 能够处理更多的观察行为；q 执行的启动行为一定是 p 能够启动执行的行为，但它选择执行的范围可能比 p 的选择范围窄小。因此，能够利用 q 替换 p，而整个系统不能感觉到替换的发生。

定义 9(进程替换)　设进程集合之上的二元关系 S，如果 S，S^{-1} 都是半替换关系，则称关系 S 为替换关系。如果 pSq，则称 q，p 可相互替换，记为 $p<>q$。

$p<>q$ 意味着进程 p，q 可以相互替换，而系统不能察觉。

4　π ADL 体系结构一致性分析

运用上面定义的相关概念，结合 π ADL 形式化描述软件体系结构的方法，下面给出 8 种 π ADL 能够执行的检测分析及其形式化定义，用以提高 π ADL 规约体系结构的质量。

(1) Port/Computation 一致性分析

构件的规约分为两个部分，即端口规约和计算部分。端口是构件行为的一个窗口，构件的计算行为必须和端口规约的相关行为一致，即是说构件必须执行端口涉及的交互行为。

设 P 是构件端口 X 的行为进程，Comp 是该构件的计算进程。如果：

$$fn(\hat{PT}_x(P)) \subseteq fn(Comp) \wedge \hat{PT}_x(P) \approx Comp/(fn(Comp)-fn(\hat{PT}_x(P)))$$

则端口 P 与构件的计算行为一致。

其中，因为计算进程中的名字用端口的名字作为前缀，故运用 $\hat{PT}_x(P)$，对端口进程 P 用端口名字 X 进行限定。端口能够执行的行为一定是计算部分执行行为的一个子集，故 $fn(\hat{PT}_x(P)) \subseteq fn(Comp)$ 必须成立。端口执行的行为模式一定是构件计算行为在端口的一个投影，故对构件计算行为中非本端口的自由名字进行隐藏，把它们作为内部行为，从而其应该与 $\hat{PT}_x(P)$ 弱等价。

(2) 连接件无死锁分析

连接件表示构件之间的交互机制，它的 Glue 表示具体的协调方式，而 Role 代表参与该协调交互机制的构件。因此，连接件规约必须确保 Glue 所描述的协调方式与 Role 所代表构件的交互行为相一致，才能够保证 Glue 正确协调各个交互构件，而不产生冲突。该种冲突能够作为进程死锁而被检测到。

设连接件的 Glue 进程为 G，其角色 R_1，R_2，…，R_n 的进程分别为 P_1，P_2，…，P_n，则：$G|\hat{PT}_{R1}(P_1)|\hat{PT}_{R2}(P_2)|\cdots|\hat{PT}_{Rn}(P_n)$ 必须无死锁。

(3) Role/Glue 控制属性分析

角色中的启动行为表示构件启动执行该行为，对连接件而言，它成为观察行为，而角色中的观察行为，一定需要连接件启动执行，角色才能够进行观察。只有这样，才能保证行为控制不发生冲突，规约才有意义。由此，得到如下检测：

设连接件的 Glue 进程为 G，R 是其任一角色，该角色的进程为 P，则：

如果 $\bar{m} \in fn(\hat{PT}_R(P))$，则 $m \in fn(G) \wedge \bar{m} \notin fn(G)$。如果 $m \in fn(\hat{PT}_R(P))$，则 $\bar{m} \in fn(G) \wedge m \notin fn(G)$。

(4)启动行为承诺分析

在 π ADL 的描述方法中，运用输出名字表达构件自主执行的启动行为。因此，构件如果能够选择执行多个启动行为，那么，该选择一定是内部选择，而不能是外部选择，即是说，构件必须承诺自己负责和控制其启动行为的执行，而不是由外部环境决定它的启动行为的执行。因此，得到如下检测：

在 πADL 规约中，设存在一个任意进程 P，它可能是构件的 Computation 进程、Port 进程、连接件的 Glue 进程、Role 进程，进程 P 必须满足如下条件：如果 $\exists s, P', P_1, P_2, \bar{e}, \bar{f}: P \overset{s}{\Rightarrow} P', P' \overset{\bar{e}}{\Rightarrow} P_1 \wedge P' \overset{\bar{f}}{\Rightarrow} P_2$

则 $P' \overset{\bar{w}}{\Rightarrow} P_1 \wedge P' \overset{\bar{v}}{\Rightarrow} P_2$。

该条件表明，在 P 执行的过程中，如果它能够对两个启动行为进行选择，则在执行选择之前，一定至少存在一次内部行为，从而表明它对启动行为所做的选择只能是内部选择。

(5) Port/Role 正确连接分析

连接件中的角色规定了什么样的端口能够连接到该角色，即连接到该角色的端口必须具备的条件。当角色和端口的行为进程彼此完全相同或者弱等价时，它们能够正确连接。但该种限制过于严格。因为构件端口描述的是该构件的具体、特定的交互方式，而连接件通常描述通用、抽象和能够在多种情况之下进行复用的交互模式，所以连接件角色和构件端口很难完全等同或行为等价。

考虑如下端口和连接件，两者明显能够正确连接，但行为却并不等价。角色 Source 表示构件能够执行任意次数的行为 $\overline{\text{Write}}$，然后发出 $\overline{\text{Close}}$ 通知并结束。

Port Out=$\overline{\text{Write}}$. Out$\prod\overline{\text{Close}}$. 0

Role Source=$\overline{\text{Write}}$. Source$\prod\overline{\text{Close}}$. 0

根据定义 5 的半替换关系定义，能够得到保证端口与角色正确连接的检测方法：

设端口的进程为 P，角色的进程为 R，如果 $R<P$ 成立，则端口 P 能够与角色 R 正确连接。

根据半替换的定义，P 能够响应所有 R 响应的行为，并且 P 只能够启动 R 所能够选择启动的行为，而且 P 可能选择启动的行为比 R 能够启动的行为少，当然运用这样的 P 代替角色，连接件不能感觉到角色被替换。

(6) 构件单向替换分析

软件体系结构静态或动态演化时，经常需要运用新的构件替换现有构件，因此必须要判定新的构件能否替换现有构件，并且保持整个系统正常运行。即要对构件进行透明的取代，被取代构件的环境不能意识构件取代的发生。根据定义，能够运用进程之间的半替换关系对此进行判定。于是存在如下检测法则：

(下转第 208 页)

过程模型与 PM_net 的转化以及基于 PM_net 的 OWL-S 过程语义一致性分析方法上。在未来的工作中，我们将重点对 PM_net 的性质进行研究，例如如何将传统着色 Petri 网的分析方法(如活性、有界性等)应用到 PM_net、如何分析 PM_net 的可靠性等。同时，我们也将进一步研究如何基于 PM_net 实现 OWL-S 过程模型的语义一致性修复，从而实现 OWL-S 本体的自动演化。

参 考 文 献

[1] McIlraith S, Son T C. Semantic Web Services[J]. IEEE Intelligent Systems, 2001(Special Issue on the Semantic Web)

[2] Martin D, Barsfin M. OWL-S: Semantic Markup for Web Services(V1.2)[Z/OL]. http://www.ai.sri.com/daml/services/owl-s/1.2, 2006

[3] Narayanan S, McIlraith S. Analysis and Simulation of Web Services[J]. Computer Networks: The International Journal of Computer and Telecommunications Networking, 2003, 42(5): 675-693

[4] 蒋运承，史忠植. OWL-S 的形式语义[J]. 计算机科学，2005，32(7)：5-7，16

[5] 李景霞，肖政，侯紫峰. 基于标签 Petri 网的 OWL-S 建模与分析[J]. 计算机工程，2007，33(7)：8-10

[6] 李景霞，侯紫峰，基于颜色 Petri 网的 Web 服务组合建模及应用[J]. 计算机应用研究，2006(9)：149-151

[7] Stojanovic L. Methods and Tools for Ontology Evolution [D]. University of Karlsruhe, 2004

[8] Haase P, Stojanovic L. Consistent Evolution of OWL Ontologies [C]// *Proceedings of the Second European Semantic Web Conference*. Heraklion, Greece, 2005

[9] 吴哲辉. Petri 网导论[M]. 北京：机械工业出版社，2006

[10] van der Aalst W, van Hee K. Workflow Management: Models, Methods, and Systems[M]. USA: MIT Press, 2002

[11] van der Aalst W M P. Verification of Workflow Nets[C]// Proceedings of the 18th International Conference on Application and Theory of Petri Nets. Berlin: Springer-Verlag, 1997

(上接第 198 页)

设构件 R 具有端口 $P_1, P_2, \cdots, P_n$，构件 S 相应地具有端口 $P_1', P_2', \cdots, P_n'$，如果：$\forall i(1 \leqslant i \leqslant n): P_i \prec P_i'$，则构件 S 能够替换构件 R，并且用端口 $P_1', P_2', \cdots, P_n'$ 替换相应端口 $P_1, P_2, \cdots, P_n$。

(7) 构件双向替换分析

在构件进行组装、交易时，两个构件的型构、行为等同样可能不同，但仍然需要经常判定它们是否能够彼此相互替换进行使用。根据定义，能够运用进程之间的替换关系对此进行判定。于是存在如下检测法则：

设构件 R 具有端口 $P_1, P_2, \cdots, P_n$，构件 S 相应地具有端口 $P_1', P_2', \cdots, P_n'$，如果：$\forall i(1 \leqslant i \leqslant n): P_i \prec\succ P_i'$，则构件 S 与构件 R 能够相互替换，端口 $P_1', P_2', \cdots, P_n'$ 与端口 $P_1, P_2, \cdots, P_n$ 同时相互对应地进行替换。

(8) 端口连接兼容分析

在实现级别，通常并没有显式的连接件，连接件已经转化成了构件或者直接运用程序语言或中间件支持的连接件。这种情况之下，通常是构件之间的端口直接进行连接，相互协调进行交互。因此，必须判定两个端口能否协调地进行交互，而不是相互冲突，从而在组装阶段可发现可能存在的错误。根据定义，能够运用进程兼容关系进行判定。于是得到如下检测：

设两个构件端口的行为进程分别是 p, q，如果 $p \exists q$，则构件端口 p 与 q 能够直接连接，两者协调进行交互。

结束语 本文基于 π ADL 软件体系结构形式化描述方法，运用 π 演算的基本理论，结合软件体系结构领域的特定需求，定义了基本的概念和进程之间的新型关系，如兼容关系、替换关系等，形式化地给出和定义了 8 种体系结构的一致性分析方法，为运用 π 演算工具进行自动化分析打下了基础。运用该 8 种分析方法，能够检查体系结构规约中出现的某些错误，提高组装软件系统的质量。

π 演算只能描述系统的行为，相关的检测和分析仅仅限于系统行为方面一致性的检测和分析。进一步的研究工作包括对 π ADL 进行扩展以描述系统的更多属性并进行更加严格的检测，如时间属性的扩展和性能分析。

参 考 文 献

[1] 梅宏，申峻嵘. 软件体系结构研究进展[J]. 软件学报，2006，17(6)：1257-1275

[2] Allen R. A *formal approach to software architecture*[D]. Pittsburgh: Carnegie Mellon University, May 1997

[3] AT&T. Best Current Practices: Software Architecture Validation[R]. AT&T. 2004

[4] Kazman R, Bass L, Abowd G, et al. SAAM: A method for analyzing the properties of software architecture[C]// Proceedings of the 16th International Conference on Software Engineering. Sorrento. Italy, May 2001: 81-90

[5] Mario R, Barbacci S, Jeromy C, et al. Steps in an architecture tradeoff analysis method: quality attribute models and analysis [R]. CMU/SEI-97-TR-029. Carnegie Mellon University, 1997

[6] Debra J R, Alexander L W. Software testing at the architecture level[C]// Proceedings of the ISAW. 2005

[7] 云晓春，方滨兴. 基于构件设计的正确性验证[J]. 小型微型计算机系统，1999，20(5)：330-334

[8] Talor R N, Medvidovic N, Anderson K M, et al. A component- and message-based architectural style for GUI software [J]. IEEE Transactions on Software Engineering, 2001, 22(6): 390-406

[9] Shaw M, DeLine R, Klein D V, et al. Abstractions for software architecture and tools to support them[J]. IEEE Transactions on Software Engineering, 2000, 21(4): 314-335

[10] Magee J, Dulay N, Eisenbach S, et al. Specifying distributed software architectures[C]// Proceedings of the Fifth European Software Engineering Conference, ESEC'95. September 1995

[11] Luckham D C, Augustin L M, Kenney J J, et al. Specification and analysis of system architecture using Rapide[J]. IEEE Transactions on Software Engineering, 1995, 21(4): 336-355

[12] 任洪敏. 基于 π 演算的软件体系结构形式化研究[D]. 上海：复旦大学，2003

[13] Milner R. Communicating and Mobile Systems: the π-Calculus [M]. Cambridge: Cambridge University Press, 1999