网构软件体系结构代数模型

赵会群*, 孙晶

北方工业大学计算机系, 北京 100041

* 通信作者. E-mail: zhaohq6625@sina.com

收稿日期: 2012-07-09; 接受日期: 2013-01-05

国家自然科学基金项目 (批准号: 61070030, 61111130121) 和北京市属高等学校人才强教深化计划学术创新团队项目 (批准号: PHR201107107) 资助

摘要 由于网络化软件的规模和复杂程度的不断增加, 设计可伸缩的软件体系结构, 开发可演化的软件系统遇到了挑战. 挑战之一是很难把软件模型与原系统说明区分开来, 使得软件系统的可演化性受到限制. 一个有效的途径是建立可范用软件体系结构的抽象模型, 指导网构软件设计, 从而提高软件模型的概括能力和可伸缩能力. 本文提出一种网构软件体系结构代数模型, 其中一个创新思想是把网构连接子解释成网构变量的运算. 结合代数学方法和软件实现技术, 定义了 6 种网构连接运算, 所有运算是连接子的代数抽象, 因此软件体系结构的抽象表达能力得到了加强. 为了展示提出代数模型的建模与分析辅助作用, 介绍了一个基于代数模型的网构软件设计与演化的案例研究.

关键词 网构软件体系结构 抽象代数 进程代数 软件演化

1 引言

随着 Internet 技术的不断发展, 基于 Internet 的网络应用也在日新月异的变化. 电子商务实现了网上购物和贸易往来^[1]; 网格计算实现了全网络计算能力的共享^[2]; 普适计算实现了无处不在的计算资源^[3]; 服务计算提供了灵活的域间业务组合机制^[4]; 云计算给用户提供了无需配置的计算和信息服务模式^[5]. 网络终端用户已经逐渐地把 Internet 看成是一个大的计算机系统, 即 “The Network is the Computer” 和 “Global Ubiquitous Computer”^[6]. 这些新的计算模式出现, 促使软件工作者对软件技术有了新的思考, 如何在这种新的 “计算机平台” 上开发、运行和维护计算机软件, 已成为计算机科学与技术面临的挑战性课题. 为了迎接这一挑战, 软件技术也经历了许多变迁. 从面向对象^[7] 到基于构件^[8] 的软件开发, 再到面向服务^[9] 的软件集成都体现出这种变化, 而这种变化也孕育了一种新的软件技术体系 “网构软件”^[10].

所谓 “网构” 是以开放、自主的方式存在于 Internet 的各个节点之上, 可在开放的环境下通过某种方式加以发布, 并以各种协同方式与其他软件实体进行跨网络的互连、互通、协作和联盟的软件实体^[10]; 而 “网构软件” 是应能感知外部环境的动态变化, 并随着这种变化按照功能指标、性能指标和可靠性指标等进行静态 (离线) 的调整和动态 (在线) 的演化, 以使系统具有尽可能高的用户满意度的一种新的软件形态. 网构软件具有自主性、协同性、反应性、演化性和多目标性等特性^[10].

引用格式: 赵会群, 孙晶. 网构软件体系结构代数模型. 中国科学: 信息科学, 2013, 43: 161-177, doi: 10.1360/112012-458

图 1 分布 ALE 模型

Figure 1 Distributed ALE Model

在网构软件开发过程中, 软件体系结构 (software architecture, SA) 扮演十分重要的角色^[11]. 然而, 由于网络化软件的规模和复杂程度的不断增加, 设计可伸缩的软件体系结构, 开发可演化的软件系统遇到了挑战. 挑战之一是很难把软件模型与原系统说明区分开来, 使得软件系统的可演化性受到限制. 一个有效的途径是建立可范用软件体系结构的抽象模型, 指导网构软件设计, 从而提高软件模型的概括能力和可伸缩能力.

结合具体应用实例, 本文从网构软件设计的需求出发, 逐层抽象, 首先建立网构软件体系结构的逻辑实体模型, 然后建立其代数模型; 在此基础上讨论网构软件的数学性质, 并把这些性质用于实际网构软件系统开发与维护中进行解释, 其目的是为网构软件技术发展提供理论上的指导和帮助, 为构建一个完整的网构软件理论体系贡献一种思路. 本文组织如下:

首先提出需求, 第 2 节通过一个案例明确网构软件体系结构建模的必要性; 第 3 节根据需求, 从网构软件自主性、协同性等 5 个方面的特性出发, 对网构软件体系结构进行第 1 次抽象, 提出网构软件体系结构的逻辑实体模型; 第 4 节在逻辑实体模型基础上, 运用进程代数和抽象代数作为建模手段, 对其进行第 2 次抽象, 提出网构软件体系结构的代数模型; 为了解释软件体系结构代数模型的实用价值, 第 5 节介绍 1 个应用案例; 文章的最后通过与同类研究工作的比较, 给出研究结论和进一步工作展望.

2 一个体系结构驱动的网络软件系统案例

本节介绍一个体系结构驱动的网络软件系统案例, 明确网构软件体系结构建模的需求, 提供系统原型和问题求解对象.

2.1 原系统描述

图 1 展示了一个 RFID (radio frequency identification) 物联网中数据报告获取的应用层事件 ALE

(application level event) 模型. 图中仅给出了 3 个识读周期, 即 Read Cycle₁ ~ Read Cycle₃. 读周期是识读者识读 EPC (electronic product code) 的最小逻辑控制单元, 它的输出是一个 EPC 集合. 一个事件周期由多个读周期构成, 按照 EPCglobal^[12] 提出的技术规范, 事件可分为 4 种不同类型: 交易事件 (transaction event)、数量事件 (quantity event)、对象事件 (object event), 以及集合事件 (aggregation event). 这里仅针对交易事件解释网构软件对体系结构建模的需求.

在图 1 所示的 ALE 模型中, 识读者可以分布在物联网不同域的不同网络节点上, 所以每一个事件生命周期都需要通过网络通信的方式提交 EPC 事件报告. 由于业务需求的不断变化, 需要不断地调整识读周期, 从而形成动态的事件生命周期. 由于物联网中识读设备的多样性, 识读者管理系统需要根据 EPC 标签的类型调用不同的识读者软件构件, 即根据识读软件变化动态配置识读者管理软件系统. 另一个动态变化来自于系统结构的演化, 图 1 所示的 ALE 模型并没有强调识读者控制的方式, 当事件周期过长而长时间占用通道, 造成交易事件事务原子性无法保证时, 可以采用非中央控制的异步通信模式, 但这可能带来比中央控制下同步模式更多的时间延迟. 为此, 可以根据具体应用场景调整其通信模式, 即构造可伸缩的软件体系结构.

2.2 网构软件体系结构建模需求

根据上述分析, 图 1 所示的 ALE 网构软件系统对体系结构有如下建模需求:

(1) 自主性. 识读者 (软件) 可以独立地工作, 虽然它需要与管理系统协调, 但它应该可以独立运行, 主动报告它的工作状态, 并根据网络环境选择适当的时机向管理系统发送采集到的业务数据, 为此需要 ALE 系统体系结构具有支持识读者构件松耦合运行的能力.

(2) 协同性. 作为软件实体, 不同识读者软件可以共同为事件周期提高业务数据, 事件周期驱动软件实体根据需要协同不同的识读者软件一同工作, 其工作模式表现出网构软件实体的协同性需求.

(3) 反应性. 反应性需求与可演化性需求相伴产生. 从业务处理角度, 由于 ALE 模型中需要根据性能需求和物联网状态动态地调整和配置通信模式, 这种通信模式的变化带来对体系结构演化能力的需求, 为此在 ALE 系统中需要部署侦听软件构件, 以便及时报告环境信息. 从软件系统实现角度, 由于识读者软件构件有更新版本的可能, 从而带来调用接口的变化, 为此 ALE 系统需要及时感知到识读者软件构件的变化.

(4) 演化性. 从处理业务和软件系统本身两个方面, 要求 ALE 体系结构有支持软件构件动态可替换能力, 以及体系结构本身的动态演化能力.

(5) 多目标性. 从 ALE 系统体系结构角度, 多目标性要求系统能提供多种组合形式, 以适应多种目标需求, 比如基于 WDSL (web services description language) 的服务形式、API (application programming interface) 函数形式、功能构件形式等, 而具有多态性的构件连接器能够实现上述形式的构件连接, 比如 BPEL (business process execution language)、远程过程调用 RFC (remote function call) 和基于消息的激发 MBI (message based invoke). 从 ALE 系统实施效果角度, 多目标性要求 ALE 可以实现一种开放的系统体系结构, 为不同系统目标服务, 比如为物流管理服务、为零售结算服务等.

3 网构软件体系结构逻辑实体模型

下面结合网构软件体系结构建模需求, 采用半形式化方法逻辑建模软件体系结构.

3.1 网构

定义 3.1 网构是一个分布在 Internet 各个节点上的软件实体^[10], 该软件实体可以是实现程序代码的语言, 可以是语言的运行环境等计算单元, 也可以是数据和数据处理结果的数据单元. 它由网构接口和网构实现模块组成. 网构接口是网构与外部环境接触点的集合, 即 $\langle \text{Port}_1, \text{Port}_2, \dots, \text{Port}_n \rangle$, 而每一个接触点 Port_i 是一个 9 元组 $\langle \text{Dom}, \text{ID}, \text{Publ}_i, \text{Refl}_i, \text{Exte}_i, \text{Priv}_i, \text{Beha}_i, \text{Mess}_i, \text{Cons}_i, \text{Non-Func}_i \rangle$. 其中:

Dom 是网构所在域, 如公司或机构名称等.

ID 是网构的标识, 是一个基于标准编码的数据结构, 可以用于安全认证、地址解析和信号位等;

Publ_i 是网构第 i 个接触点能提供给环境或其他网构的功能集合;

Refl_i 是网构第 i 个接触点能提供给环境或其他网构元数据的集合, 当元数据变化时置信号位为 1;

Exte_i 是网构第 i 个接触点运行所需环境或其他网构功能的集合, 它具有环境感知能力;

Priv_i 是网构第 i 个接触点内部属性和功能的集合;

Beha_i 是网构第 i 个接触点行为语义描述, 是一组谓词表达式;

Mess_i 是网构第 i 个接触点所产生消息的集合;

Cons_i 是对网构第 i 个接触点行为约束, 它通常包括网构运行的初始条件、前置条件和后置条件, 有时为了明确表示这 3 个条件可把它写成 $\text{Cons}(\text{init}, \text{pre-cond}, \text{post-cond})$, init , pre-cond 和 post-cond 分别表示初始条件、前置条件和后置条件的集合;

Non-Func_i 是网构第 i 个接触点非功能说明, 包括网构的服务策略、合同、安全性、可靠性说明等.

网构可以按其服务内容分类, 如可以把协调网构运行或支撑网构运行的中间件称为管理类网构, 而把提供计算能力的网构称为计算类网构等.

定义 3.2 设 A, B 是论域 Dom 中的 2 个网构, 如果 A, B 满足下列条件:

- (1) $\text{Dom}(A) = \text{Dom}(B)$,
- (2) $\text{Publ}(A) = \text{Publ}(B)$,
- (3) $\text{Refl}(A) = \text{Refl}(B)$,
- (4) $\text{Exte}(A) = \text{Exte}(B)$,
- (5) $\text{Priv}(A) = \text{Priv}(B)$,
- (6) $\text{Beha}(A) \Leftrightarrow \text{Beha}(B)$,
- (7) $\text{Mess}(A) = \text{Mess}(B)$,
- (8) $\text{Cons}(A) \Leftrightarrow \text{Cons}(B)$,
- (9) $\text{Non-Func}(A) \Leftrightarrow \text{Non-Func}(B)$,

(3.1)

则称 A, B 相等, 记作 $A = B$. ‘ $\Leftrightarrow$ ’ 为逻辑等价, ‘ $\Rightarrow$ ’ 为永真蕴涵.

3.2 网构连接子

定义 3.3 连接子是网构交互的载体, 它的逻辑实体模型是一个 6 元组 $\langle \text{ID}, \text{Role}, \text{Beha}, \text{Mess}, \text{Cons}, \text{Non-Func} \rangle$, 其中:

ID 是连接子标识, 它是一个基于标准编码的数据结构, 可以用于安全认证、地址解析和信号位等;

Role 为连接子与网构的接触点的集合, 每个 $\text{Role} = \langle \text{Id}, \text{Action}, \text{Event}, \text{LConstrains} \rangle$ 组成. 其中: Id 是 Role 的标识; Action 是 Role 活动的集合, 每个活动由事件的连接 (谓词) 组成; Event 是 Role 产

生的事件集合; LConstrains 是 Role 的约束集合. 把 Role 从连接子的其他属性分离开来的目的是突出网构的多目标性, 即一个连接子可同时与多个网构连接, 满足多种应用需求.

Mess 是连接子中各 Role 中事件产生的消息的集合;

Beha 是连接子行为的语义描述;

Cons 是连接子约束的集合, 它包括连接子的初始条件、前置条件和后置条件, 有时为了明确表示这 3 个条件可把它写成 Cons (init, pre-cond, post-cond), init, pre-cond 和 post-cond 的含义与网构中的定义相同;

Non-Func 是连接子的非功能说明, 包括按何种策略、合同、可信性连接等.

从软件实体角度, 连接子也是一种特殊的网构, 与一般的网构不同的是, 它只承担网构协同作用. 下面给出 6 种常见连接子的定义.

定义 3.4 设 A, B 是论域 Dom 中的 2 个网构, 依定义 3.1 中网构定义, 若 $\exists x \in \text{Exte}(A) \wedge \exists y \in \text{Publ}(B) \rightarrow [\text{pre-cons}(A) \wedge \text{pre-cons}(B) \wedge (x \Rightarrow y)] \wedge [\text{Mess}(x) \in \text{Mess}(B)] \wedge [\text{Mess}(y) \in \text{Mess}(A)]$, 即网构 A 通过发送一个消息“激发”组件 B 中的 $\text{Publ}(B)$ 来实现功能需求, 就称组件 A, B 进行了一次“激发”连接. 这里, $\text{Mess}(x)$ 表示活动 x 产生消息, $\text{Mess}(B)$ 表示网构 B 产生消息的集合, $\text{Mess}(x) \in \text{Mess}(B)$ 表示网构 A 发送一个功能请求, 且 B 接收活动接受了该请求; 而 $\text{Mess}(y) \in \text{Mess}(A)$ 表示构件 B 把结果返回给 A .

定义 3.5 设 A, B 是论域 Dom 中的 2 个网构, 若 $\exists x \in \text{Exte}(A) \wedge \exists y \in \text{Publ}(B)$ 使得 $[\text{pre-cond}(A) \wedge \text{pre-cond}(B) \wedge (x \Rightarrow y)] \wedge [\text{Publ}(A) = y \cup \text{Publ}(B)]$, 即网构 A 通过“使用”网构 B 中的 $\text{Publ}(B)$ 来实现功能需求, 就称网构 A, B 进行了一次“使用”连接.

定义 3.6 设 A, B 是论域 Dom 中的 2 个网构, 若 $\exists x \in \text{Publ}(A), \exists y \in \text{Publ}(B)$ 使得 $[\text{pre-cond}(A) \wedge (x)] \Leftrightarrow [\text{pre-cond}(B) \wedge (y)]$, 则称网构 B 与网构 A 协同连接.

这里 $x \Leftrightarrow y$ 并没有限定协同动作的交互方式, 所以“协同”连接依赖“使用”或“激发”连接.

定义 3.7 设 A, B 是论域 Dom 中的 2 个网构, 若 $\exists x \in \text{Publ}(A), \exists y \in \text{Publ}(B)$ 使得 $[\text{pre-cond}(A) \wedge (x)] \Leftrightarrow [\text{pre-cond}(B) \wedge (y)] \wedge [\text{pre-cond}(A) \cap \text{pre-cond}(B) = \phi]$, 则称 A 与 B 并行连接.

定义 3.8 设 A 是论域 Dom 中的一个网构, $\exists x_1 \in \text{Publ}(A), \exists y_1 \in \text{Exte}(A), \exists x_2 \in \text{Publ}(B), \exists y_2 \in \text{Exte}(B)$, 若 $[\text{post-cond}(A) \wedge \text{pre-cond}(B) \wedge (y_1 \Rightarrow x_2)] \Leftrightarrow [\text{post-cond}(B) \wedge \text{pre-cond}(A) \wedge (y_2 \Rightarrow x_1)]$, 则称 A 与 B 重复连接.

定义 3.9 设 A, B 是论域 Dom 中的 2 个网构, 若 $\exists x \in \text{Publ}(A), \exists y \in \text{Publ}(B)$ 使得 $[\text{pre-cond}(A) \wedge (x)] \vee [\text{pre-cond}(B) \wedge (y)]$, 则称 A 与 B 选择连接.

上述连接可以推广到多个网构的情形, 即 $x_{i1} op_{i1} x_{i2} op_{i2} x_{i3} op_{i3} \cdots x_{in-1} op_{in-1} x_{in}$. 其中, op_{ij} 表示上述定义的连接子, x_{ij} 表示网构.

3.3 网构软件体系结构

在明确网构、网构连接子之后, 下面给出网构软件体系结构逻辑实体模型. 首先给出体系结构风格的定义.

定义 3.10 设 $U = \{\text{Dom}_1, \text{Dom}_2, \dots, \text{Dom}_n\}$ 是网构论域的集合, 一个网构体系结构风格是网构与连接子的组合范型, 是连接子 Role 的集合.

如 C/S 结构是网构通过调用服务器中的服务构件体现其结构风格, 连接子是“使用”的实现; 而 B/S 结构是通过“激发”网构体现结构风格, 连接子是“激发”的实现. 风格还可以是多种样式的混合,

如果“使用”、“激发”, 以及其他连接子组成的综合范型.

定义 3.11 设 $U = \{\text{Dom}_1, \text{Dom}_2, \dots, \text{Dom}_n\}$ 是网构论域的集合, 一个 U 上的软件体系结构是一个三元组 $\langle C, O, \Omega \rangle$, 其中 C 为一组确定的网构的集合, O 为连接 C 中网构的连接子集合, Ω 为确定的体系结构风格.

把由定义 3.1~3.11 构建的模型称为网络软件体系结构的逻辑实体模型 (internetworkware software architecture of logic entity, ISALE), 记为: $\text{ISALE}::=\langle C, O, \Omega \rangle$, 或简称 ISALE. 其中 C 表示网构集合, O 表示网构连接子的集合, Ω 体系结构风格集合.

下面定义 ISALE 的自主性、协同性、反应性、演化性和多目标性.

定义 3.12 (自主性) 设 A, B 是论域 Dom 中的 2 个网构, OP_i 为连接子, 若 AOP_iB 满足以下特性:

- (1) $\text{Dom}(AOP_iB) = \text{Dom}(A) \cup \text{Dom}(B)$,
- (2) $\text{Publ}(AOP_iB) = \text{Publ}(A) \cup \text{Publ}(B)$,
- (3) $\text{Refl}(AOP_iB) = \text{Refl}(A) \cup \text{Refl}(B)$,
- (4) $\text{Exte}(AOP_iB) = \text{Exte}(A) \cup \text{Exte}(B)$,
- (5) $\text{Priv}(AOP_iB) = \text{Priv}(A) \cup \text{Priv}(B)$,
- (6) $\text{Beha}(AOP_iB) \Leftrightarrow \text{Beha}(A) \wedge \text{Beha}(B)$,
- (7) $\text{Mess}(AOP_iB) = \text{Mess}(A) \cup \text{Mess}(B)$,
- (8) $\text{Cons}(AOP_iB) \Leftrightarrow \text{Cons}(A) \wedge \text{Cons}(B)$,
- (9) $\text{Non-Func}(AOP_iB) \Leftrightarrow \text{Non-Func}(A) \wedge \text{Non-Func}(B)$,

(3.2)

则称 A 和 B 对连接子 OP_i 满足自主性, 若对任意一种连接子, 任意一个网构, A 与之连接均满足自主性, 则称 A 是自主的网构.

从实现技术角度, EJB 等构件技术提供了上述自主性的实现, 一个成功的案例北京大学开发的构件运行支撑平台^[13]. 因此, 网构软件自主性需求具有可行性, 而定义 3.11 也是对网构自主性的概括和刻画.

定义 3.13 (协同性) 设 A, B 是论域 Dom 中的 2 个网构, OP_i 为连接子, 若 AOP_iB 仍然是网构, 则称 A 和 B 可协同; 更进一步, 对 C 中任意的 2 个网构和 O 中的任意 1 个连接子均可协同, 则称 $\langle C, O, \Omega \rangle$ 可协同. 又设 $\langle C_1, O_1, \Omega_1 \rangle$ 和 $\langle C_2, O_2, \Omega_2 \rangle$ 是论域集合 U 中的 2 个体系结构, 如果 C_1 和 C_2 中的任意一对网构, 对 O_1 和 O_2 中的任何一个连接子均可协同, 则称 $\langle C_1, O_1, \Omega_1 \rangle$ 和 $\langle C_2, O_2, \Omega_2 \rangle$ 可协同.

从实际工程技术角度, WDLs 服务技术和 SOAP 通信技术是实现网构协同的典型技术, 因此协同性需求是可行的, 定义 3.13 是网构软件的协同性一个概括和刻画.

定义 3.14 (网构演化性) 设 A 和 B 是论域 Dom 中的 2 个网构, 如果 A 和 B 满足下列条件, 则称 B 是 A 的一个演化, 记为: $\text{Evol}(A, B)$.

- (1) $\text{Dom}(A) = \text{Dom}(B)$,
- (2) $\text{Publ}(B) \supseteq \text{Publ}(A)$,
- (3) $\text{Refl}(B) \supseteq \text{Refl}(A)$,
- (4) $\text{Exte}(B) \subseteq \text{Exte}(A)$,
- (5) $\text{Priv}(B) \subseteq \text{Priv}(A)$,
- (6) $\text{Beha}(B) \Rightarrow \text{Beha}(A)$,

(3.3)

- (7) $\text{Mess}(B) = \text{Mess}(A)$,
 (8) $(\text{Cons}(B) = \text{Cons}(A)) \text{ or } (\text{Cons}(B) \Rightarrow \text{Cons}(A))$,
 (9) $(\text{Non-Func}(B) = \text{Non-Func}(A)) \text{ or } (\text{Non-Func}(B) \Rightarrow \text{Non-Func}(A))$.

可以进一步推广到体系结构的演化.

定义 3.15 (网构体系结构演化性) 设 $\langle C_1, O_1, \Omega_1 \rangle$ 和 $\langle C_2, O_2, \Omega_2 \rangle$ 是论域集合 U 中的 2 个网构软件体系结构, 如果对 C_2 中任意一个网构 y , 总可以找到 C_1 一个网构 x , 有 $\text{Evol}(x, y)$ 成立, 并且 $O_1 \subseteq O_2, \Omega_1 \subseteq \Omega_2$. 则称 $\langle C_2, O_2, \Omega_2 \rangle$ 是 $\langle C_1, O_1, \Omega_1 \rangle$ 的一个体系结构演化, 记为 $\text{Evol}(\langle C_1, O_1, \Omega_1 \rangle, \langle C_2, O_2, \Omega_2 \rangle)$.

定义 3.16 (反应性) 设 A 和 B 是论域 U 中的 2 个网构, 如果 $\text{Evol}(A, B)$, 则称演化所需要的时间为 A 的反应性, 记为 $\text{Refl}(A, B)$; 更进一步地, 设 $\langle C_1, O_1, \Omega_1 \rangle$ 和 $\langle C_2, O_2, \Omega_2 \rangle$ 是论域 Dom 中的 2 个网构软件体系结构, 如果 $\text{Evol}(\langle C_1, O_1, \Omega_1 \rangle, \langle C_2, O_2, \Omega_2 \rangle)$, 则把 $\text{Refl}(A, B)$ 的平均时间称为 $\langle C_1, O_1, \Omega_1 \rangle$ 的反应时间, 记为: $\text{Refl}(\langle C_1, O_1, \Omega_1 \rangle, \langle C_2, O_2, \Omega_2 \rangle)$, 这里 A 是 C_1 的任意一个网构, B 是 C_2 的网构.

反应性是网构软件对环境感知能力, 以及对环境的适应能力的综合性概括和刻画. 如, 第 1 节的 ALE 结构中, 对识读器软件更新能及时感知并进行管理是对 ALE 系统的反应性需求之一.

定义 3.17 (多目标性) 设 $x_1, x_2, \dots, x_n$ 是论域 Dom 中的 n 个网构, $op_1, op_2, \dots, op_m$ 为 m 个连接子, 若 $x_{i1}op_{i1}x_{i2}op_{i2}x_{i3}op_{i3} \cdots x_{in-1}op_{im}x_{in}$ 是一个网构, 其中 x_{ij} 和 op_{ij} 分别是 n 个网构和 m 个连接子的不同排列, 则称由 $\langle C, O, \Omega \rangle$ 构成的网构软件体系结构具有多目标性, 其中 $C = \{x_1, x_2, \dots, x_n\}$, $O = \{op_1op_2 \cdots op_m\}$, $\Omega = \{\text{Role}_{ij} | \text{Role}_{ij} = x_{i1}op_{i1}x_{i2}op_{i2} \cdots x_{in-1}op_{im}x_{in}\}$.

一个技术案例是 Web Service 技术, 通过 BPEL 组合不同的服务, 实现系统多目标功能需求; ISALE 中通过连接子中的 Role 与其他属性分开来描述, 突出了网构软件的多目标性. 即一个连接子可同时与多个网构连接, 满足多种应用需求.

4 网构软件体系结构代数模型

通过把网构解释成变量、把连接子抽象为代数运算, 并针对网构软件的特性, 建立网构软件体系结构代数模型.

4.1 网构软件体系结构代数模型

ISALE 模型实现了对网构软件体系结构的逻辑抽象, 在此基础上本小节再进行数学抽象.

定义 4.1 网构变量是对 ISALE 模型中网构的代数抽象, 用大写的字母表示, 网构元素 Publ 和 Exte 中的活动称为常量.

网构变量通过值的变化贡献其对环境和其他网构变量的作用, 记通过 Publ 发布功能; 通过内部活动或者外部环境改变它的取值, 即它需要网构内部活动 Priv 和外部功能 Exte 元素; 网构变量通过运算法则展示其自身的行为, 即网构的行为 Beha 元素; 网构变量通过类型限定它的作用, 即通过类型表现它的约束. 综上所述, 网构变量可以表现网构的基本特性.

定义 4.2 设 A, B 是论域 Dom 中的 2 个网构变量, A 与 B 的激发连接、使用连接、协同连接、并行连接、重复连接、选择连接分别称为激发运算、使用运算、协同运算、并行运算、重复运算、选择运算, 记为 $\oplus, \otimes, \Theta, \parallel, \bullet$ 和 $+$.

可以证明“激发”等运算满足结合率, 而“协同”运算满足交换率, “重复”运算满足交换率, “选择”运算满足交换率. “重复”、“激发”与“使用”、“协同”与“并行”运算对“选择”运算满足分配率. 下面仅就“重复”对“选择”的分配律加以证明, 其他证明略.

定理 4.1 “重复”运算对“选择”运算满足分配律, 即

$$(A + B) \bullet C = A \bullet C + B \bullet C. \quad (4.1)$$

证明见附录 A.

定理 4.2 网构连接运算可以推广到有限个网构的情形, 即 $C_1 OP_i C_2 OP_i C_3 OP_i \cdots OP_i C_n$.

证明 根据网构运算定义, 以及网构的独立性, 定理 4.2 得证.

可以证明网构对所有运算构成代数系统.

定理 4.3 设 $\langle C, O, \Omega \rangle$ 是网构软件体系结构, 则 $\langle C, O, \Omega \rangle$ 对 O 中的每一个运算都构成代数系统.

证明 由网构连接运算的封闭性可得定理 4.3 的正确性. 证明过程参见文献 [14].

网构变量可以参与运算 (连接), 构成网构代数表达式, 把网构代数表达式称为网构软件代数模型 (internetwork software architecture algebra, ISAA), 记为 $ISAA::=\langle C, O, \Omega \rangle$. 可以证明 ISAA 有下面的基本型.

引理 4.1 任何一个程序可以只用顺序、分支和循环结构来编码 [15].

引理 4.2 任何一个使用连接均可以转化成激发连接 [16].

定理 4.4 任意一个网构软件体系结构 $\langle C, O, \Omega \rangle$ 的代数表达式 ISAA 都可以写成下面的标准型:

$$ISAA = a_1 \oplus C_1 + a_2 \oplus C_2 + \cdots + a_k \oplus C_k, \quad (4.2)$$

或简写成

$$ISAA = a_1 C_1 + a_2 C_2 + \cdots + a_k C_k, \quad (4.3)$$

其中 $a_i \in C_j$ 称为常量; $C_j \in C$, 即 $i, j = 1, \dots, K, i$ 和 j 可能不等. 证明见附录 A.

定理 4.4 的物理意义为, 一个网构软件系统可以由系统中的网构松耦合地相互激发完成系统目标. 当发生系统“循环”时, 可以通过把“循环”展开成 m 个可“选择”的“激发”来实现. 当发生“协同 (并行)”时, 由于“协同”等价于相互“激发”, 所以“协同”也可以写成“激发”的形式.

4.2 ISAA 的网构软件性质

对 ISAA 性质进行抽象, 给出 ISA 性质的代数形式描述. 首先给出网构映射的概念, 为讨论 ISA 的代数性质奠定基础.

4.2.1 网构映射与网构关系

为了更好地描述网构及其网构软件的代数性质, 下面给出几个基础定义.

定义 4.3 设 A, B 是论域 Dom 两个网构, A' 和 B' 是它们在论域 Dom 中对应的网构变量, 若 $\forall x \in \text{Publ}(A)$, 总 $\exists y \in \text{Publ}(B)$, 使得 y 与 x 对应; 同样地, 对其他元素也有同样的对应, 则称 A' 与 B' 之间存在着一个映射, 记为 $f: A' \rightarrow B'$, 或者 $B' = f(A')$.

定义 4.4 设 $ISAA_1 = \langle C_1, O_1, \Omega_1 \rangle$, $ISAA_2 = \langle C_2, O_2, \Omega_2 \rangle$, 若 $\forall x \in C_1$ 总 $\exists y \in C_2$, 使得 y 与 x 对应; 若 $\forall op_1 \in O_1$ 总 $\exists op_2 \in O_2$, 使得 op_2 与 op_1 对应; 若 $\forall m_1 \in \Omega_1$ 总 $\exists m \in \Omega_2$, 使得 m_2 与 m_1 对应, 则称 $ISAA_1$ 与 $ISAA_2$ 之间存在着一个映射, 记为 $f: ISAA_1 \rightarrow ISAA_2$, 或 $ISAA_2 = f(ISAA_1)$.

定义 4.5 设 $ISAA_1 = a_1A_1 + a_2A_2 + \cdots + a_mA_m$ 和 $ISAA_2 = b_1B_1 + a_2B_2 + \cdots + a_nB_n$, 构造一个新的网构体系结构 $ISAA_1OP_iISAA_2 = (a_1A_1OP_iISAA_2) + \cdots + (a_mA_mOP_iISAA_2) + (ISAA_1OP_ib_1B_1) + \cdots + (ISAA_1OP_ib_nB_n) + \Sigma(a_iOP_ib_j) \oplus A_iOP_iB_j$, 其中 a_i, b_j 为发生 OP_i 运算的常量对, 如“协同”运算中的一对常量、“使用”运算中的两个常量等, OP_i 表示任意一个网构运算, $\Sigma(a_iOP_ib_j) \oplus A_kOP_iB_k$ 表示多项选择. 称 $ISAA_1OP_iISAA_2$ 是 $ISAA_1$ 和 $ISAA_2$ 的构造结构, 而 $ISAA_1$ 和 $ISAA_2$ 是 $ISAA_1OP_iISAA_2$ 的因子.

在定义 4.4 的基础上可以定义 ISAA 间的关系.

定义 4.6 设 $\forall a_i \in C_1$ 和 $\forall b_j \in C_2$, 在定义 4.4 中, 所有发生 $a_iOP_ib_j$ 的 (a_i, b_j) 称为运算 OP_i 的一个关系对.

4.2.2 网构代数性质

本小节讨论网构特性的代数抽象, 为了区分网构逻辑特性, 这里称代数特性.

定义 4.7 (代数自主性) 设 A, B 是论域 Dom 中的两个网构变量, f 是 A 到 B 的映射, 若满足: (i) $f(A \oplus B) = f(A) \oplus f(B)$, (ii) $f(C_1 + C_2) = f(C_1) + f(C_2)$, 则称 A 和 B 对映射 f 满足代数自主性.

约束 (i) 和 (ii) 是对网构代码、行为、约束独立性的规约, 是对逻辑自主性的代数抽象. 只对 $\oplus$ 和 $+$ 运算进行约束是考虑到其代表性, 并且简单. 根据定理 4.3 可知, 具有自主性的网构对其他运算也满足上面的约束.

定义 4.8 (代数协同性) 设 A, B 是论域 Dom 中的 2 个网构变量, OP_i 为运算, 若 AOP_iB 封闭, 则称 A 和 B 可协同; 又设 $\langle C_1, O_1, \Omega_1 \rangle$ 和 $\langle C_2, O_2, \Omega_2 \rangle$ 是论域集合 U 中的 2 个体系结构, 如果 C_1 和 C_2 中的任意一对网构, 对 O_1 和 O_2 中的任何一个连接子均可协同, 则称 $\langle C_1, O_1, \Omega_1 \rangle$ 和 $\langle C_2, O_2, \Omega_2 \rangle$ 可协同.

代数运算封闭是对网构连接仍然是网构的抽象. 显然, 网构体系结构 $\langle C, O, \Omega \rangle$ 中的网构对所有运算应均可协同.

定理 4.5 设 $ISAA_1 = a_1A_1 + a_2A_2 + \cdots + a_mA_m$ 和 $ISAA_2 = b_1B_1 + a_2B_2 + \cdots + a_kB_k$, 则 $ISAA_1OP_iISAA_2 = (a_1A_1OP_iISAA_2) + \cdots + (a_mA_mOP_iISAA_2) + (ISAA_1OP_ib_1B_1) + \cdots + (ISAA_1OP_ib_nB_n) + \Sigma(a_iOP_ib_j) \oplus A_kOP_iB_k$, 其中 a_i, b_j 为发生 OP_i 运算的常量对, 如“协同”运算中的一对常量、“使用”运算中的两个常量等, OP_i 表示任意一个网构运算, $\Sigma(a_iOP_ib_j) \oplus A_kOP_iB_k$ 表示多项选择. 当只考虑发生关系的网构时有

$$ISAA_1OP_iISAA_2 = \sum (a_iOP_ib_j) \otimes A_kOP_iB_k. \quad (4.4)$$

证明见附录 A.

定义 4.9 (网构代数演化性) 设 A 和 B 是论域 Dom 中的 2 个网构, A', B' 是对应的网构变量, f 是 A' 到 B' 的映射. 若 $\text{Evol}(A, B)$, 则称网构变量 B' 是 A' 的代数演化, 记为 $\text{EvolA}(A', B')$.

又设 $ISAA_1 = \langle C_1, O_1, \Omega_1 \rangle$ 和 $ISAA_2 = \langle C_2, O_2, \Omega_2 \rangle$ 是论域集合 U 中的 2 个网构体系结构, f 是 $ISAA_1$ 到 $ISAA_2$ 的映射, 若对 $\forall y \in C_2$ 和 $\exists x \in C_1$, 有 $\text{EvolA}(x, y)$ 成立; 并且对 $\forall x_1 \in C_1$ 和 $\forall x_2 \in C_1$, 有 $y_1 \in C_2$ 和 $y_2 \in C_2, y_1 = f(x_1), y_2 = f(x_2)$, 使得 $f(x_1OP_ix_2) = y_1OP_jy_2$ 成立, 其中 $OP_i \in C_1, OP_j \in C_2$, 则称 $ISAA_2$ 是 $ISAA_1$ 的一个代数演化, 记为 $\text{EvolA}(ISAA_1, ISAA_2)$.

定理 4.6 任何一个网构软件体系结构 $ISAA = \langle C, O, \Omega \rangle$ 均可演化成只用调用运算 “ $\oplus$ ” 和选择运算 “ $+$ ” 连接的网构构成. 证明见附录 A.

定义 4.10 (代数反应性) 设 A 和 B 是论域 Dom 中的 2 个网构变量, 如果 $EvolA(A, B)$, 则称代数演化所需要的时间为 A 的代数学反应性, 记为 $R(A)$; 更进一步地, 设 $\langle C_1, O_1, \Omega_1 \rangle$ 和 $\langle C_2, O_2, \Omega_2 \rangle$ 是论域集合 U 中的两个网构体系结构, 如果 $EvolA(\langle C_1, O_1, \Omega_1 \rangle, \langle C_2, O_2, \Omega_2 \rangle)$, 则把 $R(A)$ 的平均时间称为 $\langle C_1, O_1, \Omega_1 \rangle$ 的反应时间, 记为 $R(\langle C_1, O_1, \Omega_1 \rangle, \langle C_2, O_2, \Omega_2 \rangle)$, 这里 A 是 C_1 的任意一个网构, B 是 C_2 的网构.

定义 4.11 (代数多目标性) 设网构体系结构 $\langle C, O, \Omega \rangle$ 具有多目标性, 则把函数 $f(x_1, x_2, \dots, x_n) = x_{i1}op_{i1}x_{i2}op_{i2}x_{i3}op_{i3}\dots x_{in-1}op_{in-1}x_{in}$ 称为目标函数.

5 网构软件体系结构代数模型的应用

本节首先建立 ALE 系统的代数模型, 然后利用代数模型分析其性质, 指导系统演化.

5.1 ALE 系统的 ISAA 模型和演化分析

按照 EPCglobal 标准设计的原 ALE 系统如图 1 所示. 然而随着业务的不断增加, 识读器和 EPC 的数量也剧烈增加. 当一个事件周期需要收集多个识读器的 EPC 数据时, 事件周期就会因长时间占用信道而引起拥塞, 从而造成交易事件的事务特性无法保证, 即长范围相关事务, 如图 1 中的事件周期 1. 为此需要对 ALE 体系结构进行演化分析.

(1) ALE 建模. 设 $Report_1$ 和 $Report_2$ 分别表示 ALE 报告生成网构; $Agent_1$ 和 $Agent_2$ 分别是 $Report_1$ 和 $Report_2$ 的代理网构; $Read_cycle_1$, $Read_cycle_2$ 和 $Read_cycle_3$ 分别表示识读器网构. 根据图 2-1 的描述, 原 ALE 系统体系结构的代数模型 $\langle C, O, \Omega \rangle$ 为

- a) $C = \{Report_1, Report_2, Agent_1, Agent_2, Read_cycle_1, Read_cycle_2, Read_cycle_3\}$;
- b) $O = \{\Theta, +\}$;
- c) $\Omega = \{Role_1 = Report_1 \oplus Agent_1, Role_2 = Agent_1 \Theta Read_cycle_1, Role_3 = Agent_1 \Theta Read_cycle_2, Role_4 = Agent_1 \Theta Read_cycle_3, Role_5 = Report_2 \oplus Agent_2, Role_6 = Agent_2 \Theta Read_cycle_2, Role_7 = Agent_2 \Theta Read_cycle_3\}$.

$ISAA_{ALE}$ 的代数表达式为

- d) $Report_1 \oplus Agent_1 \Theta (Read_cycle_1 + Read_cycle_2 + Read_cycle_3)$,
- e) $Report_2 \oplus Agent_2 \Theta (Read_cycle_2 + Read_cycle_3)$, 其中 a_1, a_2, a_3 是 $Agent_1$ 的活动; b_1, b_2 是 $Agent_2$ 的活动.

(2) 演化分析. 由广义协同性定义, 代数表达式 d) 和 e) 可以分别展开成如下形式:

- f) $Report_1 \oplus (Agent_1 \Theta Read_cycle_1 + Agent_1 \Theta Read_cycle_2 + Agent_1 \Theta Read_cycle_3 + (a_1 \Theta r_1) \oplus (Agent_1 \Theta Read_cycle_1) + (a_2 \Theta r_2) \oplus (Agent_1 \Theta Read_cycle_2) + (a_3 \Theta r_3) \oplus (Agent_1 \Theta Read_cycle_2))$;
- g) $Report_2 \oplus (Agent_2 \Theta Read_cycle_2 + Agent_2 \Theta Read_cycle_3 + (a_2 \Theta r_2) \oplus (Agent_2 \Theta Read_cycle_2) + (a_3 \Theta r_3) \oplus (Agent_2 \Theta Read_cycle_3))$;

如果考虑协同运算只在协同点上进行, 则表达式 f) 和 g) 可分别写成 f') 和 g'):

- f') $Report_1 \oplus ((a_1 \Theta r_1) \oplus (Agent_1 \Theta Read_cycle_1) + (a_2 \Theta r_2) \oplus (Agent_1 \Theta Read_cycle_2) + (a_3 \Theta r_3) \oplus (Agent_1 \Theta Read_cycle_2))$;

图 2 改进的 ALE 体系结构

Figure 2 Improved ALE architecture

$g') \text{ Report}_2 \oplus ((a_2 \Theta r_2) \oplus (\text{Agent}_2 \Theta \text{Read_cycle}_2) + (a_3 \Theta r_3) \oplus (\text{Agent}_2 \Theta \text{Read_cycle}_3))$,

其中 a_1, a_2, a_3 是 Agent_1 中的活动; r_1, r_2, r_3 是 $\text{Read_cycle}_1, \text{Read_cycle}_2, \text{Read_cycle}_3$ 中的活动; 而 $(a_1 \Theta r_1), (a_2 \Theta r_2), (a_3 \Theta r_3)$ 分别是“协同活动对”。

(3) 演化设计. 根据演化分析结果和演化的定义构造一个演化 $\emptyset$ 使得

$\emptyset(\text{Read_cycle}_1) = \text{Read_cycle}_1, \emptyset(\text{Read_cycle}_2) = \text{Read_cycle}_2, \emptyset(\text{Read_cycle}_3) = \text{Read_cycle}_3,$

$\emptyset(\text{Agent}_1) = \text{Agent}_1, \emptyset(\text{Agent}_2) = \text{Agent}_2,$

$\emptyset(\text{Report}_1) = \text{Report}_4, \emptyset(\text{Report}_2) = \text{Report}_5,$

$\emptyset(\text{Role}_1) = \emptyset(\text{Report}_1 \Theta \text{Agent}_1) = \text{Report}_4 \Theta \text{Agent}_1,$

$\emptyset(\text{Role}_2) = \emptyset(\text{Agent}_1 \Theta \text{Read_cycle}_1) = (a_1 \Theta r_1) \oplus (\text{Agent}_1 \Theta \text{Read_cycle}_1),$

$\emptyset(\text{Role}_3) = \emptyset(\text{Agent}_1 \Theta \text{Read_cycle}_2) = (a_2 \Theta r_2) \oplus (\text{Agent}_1 \Theta \text{Read_cycle}_2),$

$\emptyset(\text{Role}_4) = \emptyset(\text{Agent}_1 \Theta \text{Read_cycle}_3) = (a_3 \Theta r_3) \oplus (\text{Agent}_1 \Theta \text{Read_cycle}_3),$

$\emptyset(\text{Role}_5) = \emptyset(\text{Report}_2 \Theta \text{Agent}_2) = \text{Report}_5 \Theta \text{Agent}_2,$

$\emptyset(\text{Role}_6) = \emptyset(\text{Agent}_2 \Theta \text{Read_cycle}_2) = (a_2 \Theta r_2) \oplus (\text{Agent}_2 \Theta \text{Read_cycle}_2),$

$\emptyset(\text{Role}_7) = \emptyset(\text{Agent}_2 \Theta \text{Read_cycle}_3) = (a_3 \Theta r_3) \oplus (\text{Agent}_2 \Theta \text{Read_cycle}_3).$

为此可以采用下面两种途径解决。

(1) 采用异步通信模式. 首先把识读者捕获到的 EPC 数据单信道地接收下来, 并把数据缓冲在本地; 待一个事件周期中所有识读者的 EPC 数据全部收集完毕后, 再形成事件周期的报告. 如图 2 所示。

(2) 同步与异步动态切换的模式. 异步通信模式的代价是额外的存储, 以及相应的时间花销. 一个理想的设计或者演化途径是建立动态的体系结构, 随着环境的变化实现同步与异步的动态切换。

5.2 演化仿真实验

CADP^[17](construction and analysis of distributed processes) 是一个著名的分布式系统仿真工具集, 它不仅提供了基于进程代数的分布式系统描述语言 LOTOS^[18], 而且还提供了多种模型检测域仿真工具, 常用于软件体系结构的设计和验证.

本小节采用 LOTOS 语言分别模拟实现原 ALE 系统和演化后的 ALE 系统, 并对两者进行比较, 从而支持本文提出的方法¹⁾.

图 3 是由 CADP 提供的原系统与演化后系统的状态转换图. 一个明显的区别是原系统的状态空间大于演化后的状态空间, 所以演化后的系统在性能上有明显提高.

另一个需要考虑的问题是演化后的服务系统的可靠性情况. 为此, 我们采用与 LOTOS 相容的 μ -calculus^[19] 作为成功结束条件规约描述语言, μ -calculus 表达式如下:

$$[\text{true} * \text{'REQUEST!1!2!3'}.(\text{not 'REPORT!1!2!3'})^*] < (\text{not 'REPORT!1!2!3'})^* \text{'REPORT!1!2!3'} > \text{true}.$$

它的语义为: 一个报告订阅者能够获得 EPC 数据报告.

检测结果如图 4 所示, 原服务系统有反例出现, 即无法产生 EPC 数据报告. 原因是在非协同点上, 由于没有协同活动对进行正常交互, 所以有故障发生. 而演化后的 ALE 系统只在协同点上交互, 因此没有失败发生.

表 1 是图 3 和 4 的统计结果. 包括演化前后状态空间的状态数、转移次数和执行时间的统计; 演化前检查出错误时状态节点数、转移次数和时间; 演化后成功时的状态节点数、转移次数和执行时间.

6 相关研究比较及结论

软件体系结构建模相关研究已经有近 20 年的历史, 文献 [20~25] 分别从不同角度综述了研究现状, 并提出了新的研究方向. 值得关注的是这些文献都把动态、可信的 SA 建模与检测作为未来的研究重点, 近期的研究成果也表明了这一观察的准确性. 侯丽珊等^[26] 基于网构软件模型, 提出用软件与环境相互作用刻画软件行为的方法, 给出了基于环境变迁的网构软件组合算法模型. 李长云等^[27] 基于高阶多型 π 演算理论, 提出了动态体系结构描述语言 D-ADL. 周立等^[28] 针对网构软件行为中的不确定性和不完整性, 提出了一种支持协商的网构软件体系结构行为建模与验证方法; 通过扩展 UML 建模元素支持行为的不确定与不完整建模; 提出基于模型检验 Spin 反例引导的正确性验证方法. 常志明等^[29] 基于 Bigraph 理论对自适应 SA 进行了形式化规约, 分析和验证了系统动态演化的性质, 并探讨了 Bigraph 理论在自适应 SA 形式化方面的优势和拓展方向. 丁博等^[30] 提出了一种面向普适计算的 SA 风格 UbiArch, 设计实现了支撑该风格的软件平台原型系统. 沈立炜等^[31] 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, 并提出了面向方面机制的体系结构描述语言和原型工具. Medvidovic^[32] 在肯定了 UML (unified modeling language) 的能力后, 给出了 UML 与 ADL (architecture description language) 相互缩主的两种建模策略. 但作者经过大量的案例分析后, 并没有给出 UML 在建模 SA 方面的肯定, 而是指出了诸多不足, 如 UML 在结构风格、连接器描述、局部和全局约束等. Malavolta 等^[33] 针对 15 年来 ADL 研究和应用中出现的符号不统一、又各有其特点的复杂局面, 提出一个 ADL 及其工具的交互框架 DUALY, 它可以实现不同 ADLs 和相应工具之间的

1) 原系统的 LOTOS 描述见电子附件 (11202-458-code). 该附件未经编辑或排版, 其内容的科学性、准确性由作者文责自负.

图 3 (a) 原服务系统状态转换图; (b) 改进后的服务系统状态转换图

Figure 3 (a) State transit chart of original system; (b) state transit chart of improved system

图 4 (a) 原 ALE 系统失败反例图; (b) 演化后 ALE 系统业务成功状态空间

Figure 4 (a) State chart of original ALE system with fails; (b) state chart of normal ALE system

表 1 演化前后对比数据表

Table 1 Complexity contrast of state space fore and aft evolution

	States	Transits	Perform time (ms)	Result
Original	24	34	42	true
Improved	8	14	48	true
Fail	10	12	36	false
Pass	20	37	64	true

映射. Bucchiarone 和 Galeotti^[34] 采用图语法建模软件体系结构, 并特别关注软件体系结构的变化; 通过按类划分软件体系结构的变化, 并用图语法进行描述, 从而方便体系结构演化的建模与检验, 使

用工具 DYNALLOY 设计实现, 取得了满意的效果. Mens 等^[35] 在强调软件体系结构在软件开发中的主导地方后, 进一步论述了软件体系结构建模语言的演化的必要性, 通过案例解释了如果改进体系结构的描述语言.

上述研究主要采用形式化或半形式化的方法建模软件体系结构, 提出软件体系结构的描述语言, 如进程代数、 π 演算, 以及相应的 ADLs^[24]. 但由于这些形式化技术并非为计算机软件建模所设计, 所以在描述 SA 时必须进行扩展和改编, 描述能力不免受到影响. UML 虽然可以直接用于程序设计, 但又存在对 SA 表述能力的不足等问题^[25]. 不仅如此, 一个被长期忽视的问题是 SA 建模方法强烈地依赖于软件需求说明, 这样就会造成软件体系结构要不断地适应软件需求的变更的问题. 本文提出的网构软件体系结构代数模型, 基于软件需求但进行了必要的数学抽象, 在更抽象的层面上讨论体系结构模型和性质, 使得模型的概括能力得到增强, 可以指导一类软件设计与分析中的问题, 如网构软件可达性判定问题、基于软件体系结构可信范式的软件检验等. 作者在软件体系结构代数模型方面工作多年, 形成的一些成果也说明这种途径的可行性^[14,16,36].

本文针对网构软件体系结构建模需求, 结合一个具体案例对系统体系结构进行了两级抽象, 即逻辑抽象和数学抽象, 最终提出了一个网构软件体系结构的代数模型. 在此基础上用数学方法进行演绎, 归纳出软件体系结构的代数性质, 把这些性质用于实际软件的建模和分析中, 取得一定的效果. 具体贡献如下:

- 1) 提出了网构软件体系结构逻辑实体模型, 在逻辑层面上给出网构软件自主性、协同性、反应性、演化性和多目标性定义, 为建立代数模型奠定了基础.
- 2) 提出了网构软件的代数模型, 一个创新的思想是把网构连接子解释成网构运算, 通过证明其代数性质后, 建立了网构软件体系结构的代数模型; 进一步对自主性等网构软件的性质建模, 从而建立了代数性质到网构软件实现的对应关系, 指导网构软件设计与分析.
- 3) 通过代数抽象建立的网构软件体系结构模型, 基于软件需求但进行了必要的数学抽象, 在更抽象的层面上讨论体系结构模型和性质, 使得模型的概括能力得到增强, 可以指导一类软件设计与分析中的问题.

用代数学方法建模网构软件体系结构虽然不是全新的尝试, 但目前还没有形成完整的理论体系和相关工具的支持. 为此, 我们将在以下几个方面继续开展工作.

- 1) 进一步丰富网构软件建模理论, 研究可信网构软件体系结构的建模方法和评价方法;
- 2) 明确代数模型与传统代数学的联系, 尝试用代数学方法解决网构软件设计与分析中的复杂求解问题;
- 3) 设计和实现支持代数模型建模的工具, 开发各种应用案例, 支持提出的建模理论.

参考文献

- 1 Bonnett K R. An IBM Guide to Doing Business on the Internet. McGraw-Hill, 2000
- 2 Foster I, Kesselman C. The Globus project: A status report. In: Heterogeneous Computing Workshop (HCW), Oriando, 1998. 4-18
- 3 Marco M, Franco Z. Programming pervasive and mobile computing applications: the TOTA approach. ACM Trans Softw Engin Method, 2009, 18: Article 15
- 4 Huhns M, Singh M P. Service-oriented computing: Key concepts and principles. IEEE Internet Comput, 2005, 9: 75-81
- 5 Zhang L J, Zhou Q. CCOA: Cloud computing open architecture. In: International Conference on Web Seivces, ICWS, 2009. 607-616

- 6 Milner R. Theories for the global ubiquitous computer. In: Foundations of Software Science and Computation Structures, LNCS 2987. Berlin: Springer-Verlag, 2004
- 7 Booch G. Object-Oriented Analysis and Design with Applications. Reading: Addison-Wesley, 1994
- 8 Bachman F, Bass L, Buhman C. Technical Concepts of Component-Based Software Engineering. CMU/SEI-2000-TR-008, ESC-TR-2000-007. USA, 2000
- 9 Michael S. Using architectural patterns and blueprints for service-oriented architecture. IEEE Softw, 2006, 23: 54–61
- 10 Yang F Q, Mei H, Lv J, et al. Some discussion on the development of software technology. Acta Electron Sin, 2002, 30: 1901–1906 [杨芙清, 梅宏, 吕建, 等. 浅论软件技术发展, 电子学报, 2002, 30: 1901–1906]
- 11 杨芙清, 吕建, 梅宏. 网构软件技术体系: 一种以体系结构为中心的途径. 中国科学 E 辑: 信息科学, 2008, 38: 818–828
- 12 EPCglobal. The Application Level Events (ALE) Specification, Version 1.0. EPCglobal Ratified Standard. September 2005. http://www.epcglobalinc.org/standards/ale/ale_1.0_standard_20050915.pdf
- 13 Huang G, Wang Q X, Cao D G, et al. PKUAS: A domain-oriented component operating platform. Acta Electron Sin, 2002, 30: 39–43 [黄罡, 王千祥, 曹东刚, 等. 一种面向领域的构件运行支撑平台, 电子学报, 2002, 30: 39–43]
- 14 Zhao H Q, Sun J. An algebraic model of service oriented trustworthy software architecture. Chinese J Comput, 2010, 33: 890–900 [赵会群, 孙晶. 面向服务的可信软件体系结构代数模型. 计算机学报, 2010, 33: 890–900]
- 15 Bohm C, Jacopini G. Flow diagram, turing machine and languages with two formation rules. In: CACM. New York: ACM Press, 1966. 645–653
- 16 Zhao H Q, Wang G R, Gao Y. An abstract model of software architecture. Chinese J Comput, 2002, 25: 730–736 [赵会群, 高远. 软件体系结构抽象模型. 计算机学报, 2002, 25: 730–736]
- 17 Garavel H, Lang F, Mateescu R, et al. CADP 2010: A toolbox for the construction and analysis of distributed processes. In: TACAS, Saarbrücken, 2011
- 18 Bolognesi T, Brinksma E. Introduction to the ISO specification language LOTOS. Comput Netw, 1987, 14: 25–59
- 19 Mateescu R. Local model-checking of modal mu-calculus on acyclic labelled transition systems. In: Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), Grenoble, 2002
- 20 Mei H, Shen J R. Progress of Research on Software Architecture. J Softw, 2006, 17: 1257–1275 [梅宏, 申峻嵘. 软件体系结构研究进展. 软件学报, 2006, 17: 1257–1275]
- 21 吕建, 马晓星, 陶先平, 等. 网构软件的研究与进展. 中国科学 E 辑: 信息科学, 2006, 36: 1037–1080
- 22 Sun C A, Jin M Z, Liu C. Overviews on software architecture research. J Softw, 2002 13: 1228–1237 [孙昌爱, 金茂忠, 刘超. 软件体系结构研究综述. 软件学报, 2002, 13: 1228–1237]
- 23 Zhang P C, Li B X, Zhou Y. Reserch and development of model checking software architecture. Comput Sci, 2007, 34: 7–12 [张鹏程, 李必信, 周宇. 模型检测软件体系结构研究与进展. 计算机科学, 2007, 34: 7–12]
- 24 Medvidovic N, Taylor R N. A classification and comparison framework for software architecture description languages. IEEE Trans Softw Engin, 2000, 26: 156–168
- 25 Shaw M, Clements P. The Golden Age of Software Architecture: A Comprehensive Survey. Technical Report CMU-ISRI-06-101. Carnegie Mellon Univ, Feb. 2006
- 26 Hou L S, Jin Z. A composition model of internet components based on environment transitions. J Nanjing Univ, 2005, 41: 174–179 [侯丽珊, 金芝. 基于环境变迁的构件组合模型. 南京大学学报, 2005, 41: 174–179]
- 27 Li C Y, Li G S, He P J. A formal dynamic architecture description language. J Softw, 2006, 17: 1349–1359 [李长云, 李赣生, 何频捷. 一种形式化的动态体系结构描述语言. 软件学报, 2006, 17: 1349–1359]
- 28 Zhou L, Chen X P, Huang G, et al. Negotiation-enabled modeling and verification of architectural behavior of inter-netware. J Softw, 2008, 19: 1009–1112 [周立, 陈湘萍, 黄罡, 等. 支持协商的网构软件体系结构行为建模与验证. 软件学报, 2008, 19: 1009–1112]
- 29 Chang Z M, Mao X J, Qi Z C. Applying bigraph theory to self-adaptive software architecture. Chin J Comput, 2009, 32: 97–106 [常志明, 毛新军, 齐治昌. Bigraph 理论在自适应软件体系结构上的应用. 计算机学报, 2009, 32: 97–106]
- 30 Ding B, Wang H M, Shi D X. An adaptive software architecture style for pervasive computing. J Softw, 2009, 20(Suppl): 113–122 [丁博, 王怀民, 史殿习. 一种面向普适计算的适应性软件体系结构风格. 软件学报, 2009, 20(增): 113–122]
- 31 Shen L W, Peng X, Zhao W Y. Software product line architecture modeling and component composition implementation with extension of aspectual mechanism. Electron Sin, 2009, 37: 140–145 [沈立伟, 彭鑫, 赵文耘. 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. 电子学报, 2009, 37: 140–145]
- 32 Medvidovic N. Modeling software architectures in the unified modeling language. ACM Trans Softw Engin Method, 2002, 11: 2–57
- 33 Malavolta I, Muccini H, Pelliccione P, et al. Providing architectural languages and tools interoperability through model transformation technologies. IEEE Trans Softw Engin, 2010, 36: 119–140

- 34 Bucchiarone A, Galeotti J P. Dynamic software architecture verification using DynAlloy. In: Electronic Communication of the European Association of Software Science and Technology, Vol. 10, Graph Transformation and Visual Modelling, 2008
- 35 Mens T, Magee J, Rumpe B. Evolving software architecture descriptions of critical systems. IEEE Comput, 2010, 43: 42–48
- 36 Zhao H Q, Sun J, Wei Y, et al. Study on the method of checking the reachability of internet ware based on model. Chin J Comput, 2011, 34: 1001–1011 [赵会群, 孙晶, 魏莹, 等. 基于模型的网构软件可达性检测方法研究. 计算机学报, 2011, 34: 1001–1011]

附录 A 部分定理证明过程

定理 4.1 证明 要证明 (4.1) 式成立, 只要证明等式的左边和等式右边满足定义 3.2 中的 9 个条件即可. 为此, 我们有代表性的证明第 2 个和第 6 个条件成立, 其他证明同理.

首先, 对 $\bullet$ 和 $+$ 运算, 都有 $\text{Publ}(AOP_i B) = \text{Publ}(A) \cup \text{Publ}(B)$ 性质, 所以对 $\forall x \in \text{Publ}((A+B) \bullet C)$, 有 $x \in \text{Publ}(A) \vee x \in \text{Publ}(B) \vee x \in \text{Publ}(C)$, 可得 $x \in \text{Publ}(A \bullet C + B \bullet C)$. 同理, 对 $\forall y \in \text{Publ}(A \bullet C + B \bullet C)$ 有 $y \in \text{Publ}((A+B) \bullet C)$. 因此, 有 $\text{Publ}((A+B) \bullet C) = \text{Publ}(A \bullet C + B \bullet C)$, 定义 3.2 的性质 (2) 成立.

首先, 对 $\bullet$ 运算, 有 $\text{Beha}(A \bullet B) \Leftrightarrow \text{Beha}(A) \wedge \text{Beha}(B)$ 性质; 而对 $+$ 运算, 有 $\text{Beha}(A+B) \Leftrightarrow \text{Beha}(A) \vee \text{Beha}(B)$ 性质. 由 $\text{Beha}((A+B) \bullet C) \Leftrightarrow (\text{Beha}(A+B) \wedge \text{Beha}(C)) \Leftrightarrow (\text{Beha}(A) \vee \text{Beha}(B)) \wedge \text{Beha}(C) \Leftrightarrow (\text{Beha}(A) \wedge \text{Beha}(C)) \vee (\text{Beha}(B) \wedge \text{Beha}(C)) \Leftrightarrow \text{Beha}(A \bullet C + B \bullet C)$, 则 $\text{Beha}((A+B) \bullet C) \Leftrightarrow \text{Beha}(A \bullet C + B \bullet C)$, 性质 (6) 成立. 证毕.

定理 4.4 证明 用归纳法证明假设 $k=0$ 时成立, 此时 ISAA 只有一个网构构成, $\text{ISAA} = a_1 C_1$ 表示该组件的启动、运行与反馈的状态转换.

现假设 $K=n-1$ 时成立, 去证明 $K=n$ 时成立, 即 $\text{ISAA} = a_1 C_1 + a_2 C_2 + \cdots + a_{n-1} C_{n-1}$ ($a_i \in C_j$; $C_j \in C$, 即 $i, j = 1 \cdots K, i, j$ 可能不等) 成立, 去证明 $\text{ISAA} = a_1 C_1 + a_2 C_2 + \cdots + a_{n-1} C_{n-1} + a_n C_n$.

(1) “使用” 运算由引理 4.2 显然成立.

(2) “选择” 运算时显然成立.

$$\text{ISAA} = a_1 C_1 + a_2 C_2 + \cdots + a_{n-1} C_{n-1} + a_n C_n. \quad // \text{设 } a_n \in C_j, C_n \in C$$

(3) 证明 “重复” 运算成立. 设 $a_n \in C$, 有 $\text{ISAA} = (a_1 C_1 + a_2 C_2 + \cdots + a_{n-1} C_{n-1}) a_n C_n$; 由于由 “ $\bullet$ ” 满足对 “ $+$ ” 的分配率, 则 $\text{ISAA} = (a_1 C_1 a_n C_n + a_2 C_2 a_n C_n + \cdots + a_{n-1} C_{n-1} a_n C_n)$; 设 $x_i \in \text{Publ}(C_i)$, 则 $\text{ISAA} = (a_1 x_1 a_n C_n + a_2 x_2 a_n C_n + \cdots + a_{n-1} x_{n-1} a_n C_n + C)$; 构造 $b_i = a_i x_i a_n$, $b_n C_n = C$ (一个空调用), 则 $\text{ISAA} = b_1 C_1 + b_2 C_2 + \cdots + b_{n-1} C_{n-1} + b_n C_n$, 即定理在 “重复” 运算下成立.

(4) 证明 “顺序” 运算成立. 设 $a_n \in C, C_n \in C$, 有 $\text{ISAA} = (a_1 C_1 + a_2 C_2 + \cdots + a_{n-1} C_{n-1}) \oplus a_n C_n$; 由 $\oplus$ 对 $+$ 运算满足分配率, 有 $\text{ISAA} = (a_1 C_1 \oplus a_n C_n + a_2 C_2 \oplus a_n C_n + \cdots + a_{n-1} C_{n-1} \oplus a_n C_n)$; 设 $x_i \in \text{Publ}(C_i)$, 则 $\text{ISAA} = (a_1 x_1 \oplus a_n C_n + a_2 x_2 \oplus a_n C_n + \cdots + a_{n-1} x_{n-1} \oplus a_n C_n + C)$; 构造 $b_i = a_i x_i \oplus a_n$, $b_n C_n = C$ (一个空调用), 则 $\text{ISAA} = b_1 C_1 + b_2 C_2 + \cdots + b_{n-1} C_{n-1} + b_n C_n$, 即定理在 “顺序” 运算下成立.

由引理 4.1 得定理 4.4 得证.

定理 4.5 证明 由积结构定义可知: $\text{ISAA}_1 OP_i \text{ISAA}_2 = (a_1 A_1 OP_i b_j B_j) + \cdots + (a_m A_m OP_i b_j B_j) + (a_i A_i OP_i b_1 B_1) + \cdots + (a_i A_i OP_i b_n B_n) + \Sigma(a_i OP_i b_j) \oplus A_i OP_i B_j = (a_i A_i OP_i b_j B_j) + (a_i A_i OP_i b_j B_j) + \Sigma(a_i OP_i b_j) \oplus A_i OP_i B_j$. 考虑任何一个 $(a_i A_i OP_i b_j B_j)$, 由于只考虑 a_i 和 b_j 为发生 OP_i 运算的常量, $a_i A_i OP_i b_j B_j = a_i \varepsilon OP_i b_j \varepsilon = a_i OP_i b_j$ (这里的 ε 是网构的空活动), 而 $a_i OP_i b_j$ 在多项选择式中已经被列入, 所以它应该被削掉, 即有 (4.4) 式成立.

定理 4.6 证明 构造一个映射 $\emptyset$ 使得:

a) ISAA 中的每一个网构映射到它本身;

b) $\emptyset(C_1 \oplus C_2) = \emptyset(C_1) \oplus \emptyset(C_2)$, $\emptyset(C_1 + C_2) = \emptyset(C_1) + \emptyset(C_2)$ (网构自主性条件).

只需要对重复运算 “ $\bullet$ ” 和协同运算 “ Θ ” 给出证明.

1) 对重复 “ $\bullet$ ” 运算. 根据重复运算的定义, 有 $[(\text{post-cond}(y_1) \Rightarrow \text{pre-cond}(x_2))] \Leftrightarrow [\text{post-cond}(y_2) \Rightarrow \text{pre-cond}(x_1)]$, 其中 $\exists x_1 \in \text{Publ}(C_1), \exists y_1 \in \text{Exte}(C_1), \exists x_2 \in \text{Publ}(C_2), \exists y_2 \in \text{Exte}(C_2)$; 又由于调用运算均需要满足条件: $\text{post-cond}(A) \Rightarrow \text{pre-cond}(B)$, 所以重复运算 “ $\bullet$ ” 可以用 $C_1 \oplus C_2 \oplus C_1 \oplus \cdots \oplus C_1 \oplus C_2 \oplus C_1 + C_2 \oplus C_1 \oplus C_2 \oplus \cdots \oplus C_2 \oplus C_1 \oplus C_2$ 共 n 个调用选择项替换, n 为网构重复调用次数. 再考虑映射 $\emptyset$.

$\emptyset(C_1 C_2) = \emptyset(C_1 \oplus C_2 \oplus C_1 \oplus \cdots \oplus C_1 \oplus C_2 \oplus C_1 + C_2 \oplus C_1 \oplus C_2 \oplus \cdots \oplus C_2 \oplus C_1 \oplus C_2) = \emptyset(C_1) \oplus \emptyset(C_2) \oplus \emptyset(C_1) \oplus \cdots \oplus \emptyset(C_1) \oplus \emptyset(C_2) \oplus \emptyset(C_1) \oplus \emptyset(C_2) \oplus \cdots \oplus \emptyset(C_2) \oplus \emptyset(C_1) \oplus \emptyset(C_2) = \emptyset(C_1) \emptyset(C_2)$. 因此, 对重复运算 “ $\bullet$ ” 是一个演化, 而且重复运算 “ $\bullet$ ” 可以演化成 n 个调用选择项.

2) 对协同运算 “ Θ ”. 根据协同运算的定义, $[(\text{pre-cond}(y_2)) \Rightarrow (\text{pre-cond}(x_1))]$ 与 $[(\text{pre-cond}(y_1)) \Rightarrow (\text{pre-cond}(x_2))]$ 同时成立, $H = \text{pre-cond}(C_1) \cap \text{pre-cond}(C_2) \neq \phi$ 是同步条件集, 其中 $\exists x_1 \in \text{Publ}(C_1)$ 、 $\exists y_1 \in \text{Exte}(C_1)$ 、 $\exists x_2 \in \text{Publ}(C_2)$ 、 $\exists y_2 \in \text{Exte}(C_2)$. 又根据调用运算需要满足的条件: $(\text{pre-cond}(x)) \Rightarrow (\text{pre-cond}(y))$, 则一个协同运算可以用等价于一对调用选择项 $C_1 \oplus C_2 \oplus C_1 + C_2 \oplus C_1 \oplus C_2$. 再考虑映射 $\emptyset$:

$\emptyset(C_1 \Theta C_2) = \emptyset(C_1 \oplus C_2 \oplus C_1 + C_2 \oplus C_1 \oplus C_2) = \emptyset(C_1) \oplus \emptyset(C_2) \oplus \emptyset(C_1) \oplus \emptyset(C_2) \oplus \emptyset(C_1) \oplus \emptyset(C_2) = \emptyset(C_1) \Theta \emptyset(C_2)$. 因此, 对协同运算 “ Θ ” 是一个演化, 而且重复运算 “ Θ ” 可以演化成 2 个调用选择项.

An algebraic model of Internetware software architecture

ZHAO HuiQun* & SUN Jing

Department of Computer Science, North China University of Technology, Beijing 100041, China

*E-mail: zhaohq6625@sina.com

Abstract As the complexity and the dimensions of Internet software systems have been increasing dramatically, the design of scalable software architecture and the development of high quality program have become an urgent and inevitable problem. The big challenge lying in the development cycle is that it is difficult to clearly differentiate a software model with its original system specification. An effective approach to solving this problem is to define a mathematical model for software architecture, which will unambiguously abstract the architecture of the software from an original system specification. In this paper, we propose an algebraic model for Internetware software architecture (ISA), a novel idea for constructing ISA algebra model is that the connectors are interpreted as an implement of algebra operator. Six operators in total in this paper are given with respect to process algebra and programming technology. All the operators get its semantic from both the algebra operators and invoke sentences so that both the system specification and the software model are clearly differentiated in the proposed ISA model. One typical application has been presented as case studies for the evolution design and simulation of an IoTs application.

Keywords Internetware software architecture, abstract algebra, process algebra, software evolution

ZHAO HuiQun was born in 1960 in Shenyang, Liaoning Province. He received B.S. in mathematics in 1983, M.S. in computer science in 1991, and Ph.D. in computer science in 2002 from Northeast University, China. He was a research postdoctoral fellow from 2002 to 2004 at East Software Corporation.

He is currently a Professor of Computer Science at North China University of Technology. His research interests include dependable software architecture, Internetware and software testing. He is team leader of software engineering and Internet of Things from North University of Technology, which have been supported by National Natural Science Foundation of China and Innovative Scientific Research Team of Beijing Education Committee respectively.

SUN Jing was born in 1968 in Tieling, Liaoning Province and received B.S. in information science in 1988, M.S. degree in computer science in 2002 from LiaoNing University. She is an Associate Professor of computer science at North China University of Technology. Her research interests are Internetware, software testing and Internet of Things. She is a key member of software engineering team from North University of Technology.