构件演化中的系统行为一致性的研究*)

罗 毅 李兴宇 关连伟 胡 昊 吕 建

(南京大学计算机软件新技术国家重点实验室 南京大学计算机软件研究所 南京 210093)

摘 要 构件技术的发展,减少了开发应用程序的时间和复杂度,同时也为软件提供了更好的动态演化能力。基于构件的软件系统是通过在构件间建立交互关系,将多个构件组织成一个统一的整体得到的。因此在构件演化时,例如对构件功能实现进行改变,可能导致系统运行偏离原来的系统行为。我们的工作就是在构件演化时对系统行为进行一致性检查,保证构件的功能实现的变化不会使系统行为偏离原来的系统。在本文中首先通过 Petri-net 的形式化方法,对系统实现中包括的构件的功能实现和构件间的交互进行建模,并通过以上信息推导得到系统行为。在此基础之上,根据基于行为继承理论的行为一致性规则的要求对定义的系统功能行为进行验证,以保证构件演化时系统行为的改变符合行为的一致性要求,同时又保证了构件演化的灵活性。

关键词 构件演化,系统行为,行为一致性

Study on Behavior Consistency of System on Component Evolution

LUO Yi LI Xing-Yu GUANG Lian-Wei HU Hao LU Jian

(State Key Laboratory for Novel Software Technology, Institute of Computer Software, Nanjing University, Nanjing 210093)

Abstract Component based software development (CBSD) facilitate the construction of software and support dynamic evolution. In CBSD, software is built by assembling components which are already developed and prepared for integration, so the software system behavior is changed with component evolution. Our work is to verify system behavior and ensure behavior consistency of system. At first, component behavior and interaction between components are formalized by Petri-net, and then, system behavior is verified by consistency rule based on behavior inheritance theory. In this way, system behavior is consistent when component is evolving.

Keywords Component evolution, System behavior, Behavior consistency

1 引言

随着构件技术的快速发展,越来越多的系统利用构件来开发大型复杂软件系统[2]。通常意义上来说,基于构件的软件开发是集成多个构件来构造软件系统,构件的集成就是在构件间建立交互关系,协调它们的行为,把它们组织成一个有机的整体[1]。由于在基于构件的软件开发中有区分构件的提供者和系统装配者,因此基于构件的软件开发不可避免地存在构件的替换和更新,我们称之为构件演化[3]。而在构件演化时,构件本身的功能实现的变化可能影响系统行为,导致系统行为的改变,这是因为系统行为是由构件的内部状态变迁和构件间的交互共同作用形成的整体的功能执行序列[4]。不受约束的系统行为改变可能导致系统出错,因此在演化时引入某些约束条件对于系统正确运行有着很重要的意义。在这里,我们把通过保留系统行为的特征来保证系统运行正确性的约束条件称为一致性约束规则。

对于在构件演化时保证系统行为一致性是现在的研究热点,文[8]和[12]通过在新构件和原来的构件间建立相容性和替换性,通过局部构件行为的一致性来实现全局的系统行为的一致性,保证原来系统环境下对新构件使用不会导致系统行为出错。而在本文中则通过构件的功能行为和构件间的交互行为合成系统行为,并在系统行为层面而不是在构件行为层面加入一致性约束,来保证系统的功能行为的延续。在新的系统的功能行为是延续了原来系统功能行为的情况下,系统行为执行的正确性就可以得到保证。这样,不需要对替换的构件行为进行严格的行为约束,又保证了运行时的系统行为不会因此偏离原来的系统而产生错误。

在本文中采用了基于 Petri-net 的模型来形式化构件的行为和构件间的交互行为,并借鉴了 Ebert 和 Engles 为定义类及其子类行为描述的相互关系而提出的对象行为和相应的一致性关系:观察一致性[13]。在这个规则的约束下,能保证新系统行为中执行的功能序列是继承自原来的系统,可以保证系统行为不会偏离原来的系统,避免系统错误的产生。

本文以下部分的组织如下:第 2 部分介绍构件演化时改变构件功能实现对系统行为的影响以及产生的问题;第 3 部分给出系统实例的描叙,其中包括构件行为、构件间交互行为的形式化定义,并给出了在此基础之上生成系统行为的过程和算法;第 3 部分定义一致性检查的约束规则以及使用的场景;第 4 部分介绍基于 OSGi 平台的系统实现,即支持系统行为动态监控和检查的集成应用;最后讨论相关工作和进一步

*)本项目受国家 863 计划(2004AA112090,2005AA113160,2005AA113030)、国家 973 计划(2002CB312002)、国家自然科学基金(60273034,60233010,60403014)资助。**罗 毅** 硕士研究生,主要研究领域为软件工程、软件体系结构;**李兴宇** 硕士研究生,主要研究领域为软件工程;**关连伟** 硕士研究生,主要研究领域为软件过程建模;**胡 昊** 讲师,主要研究领域为软件过程、工作流技术、移动 Agent 技术;**吕 建** 教授,博士生导师,主要研究领域为对象技术、分布计算技术、移动 agent 技术。

的工作。

2 构件演化时系统行为不一致性问题

通常情况下，软件构件在进行演化时，构件本身功能实现的改变可能影响系统行为，导致系统功能行为执行的改变。而这种改变是有可能导致系统行为偏离系统的功能目标，产生错误的。本文首先通过一个系统实例来介绍在构件间交互不变的情况下，构件功能实现改变时，对系统行为产生影响所引发的问题，来研究如何在构件的功能实现改变，保证系统功能行为的改变不会偏离原有系统。

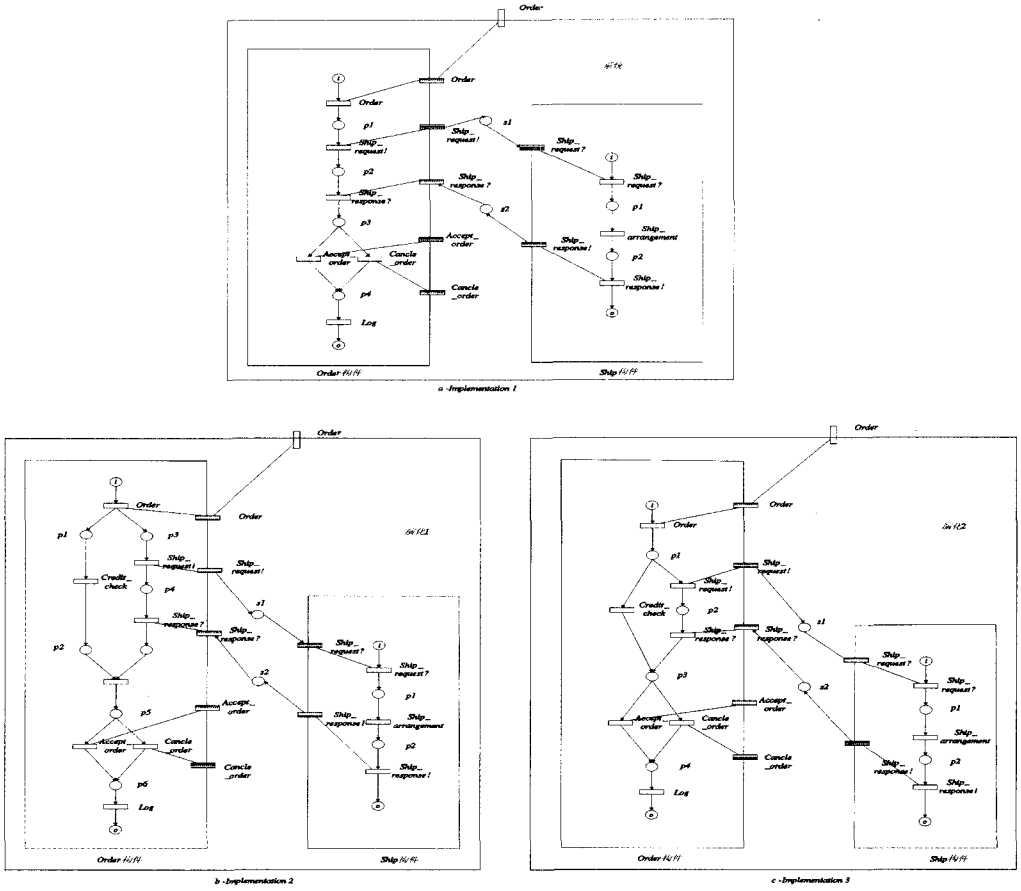

图 1 一个订货系统的具体实现

图 1(a)显示了一个订货系统的具体实现。系统由两个构件组成：构件 Order 和构件 Ship。构件 Order 通过 Order 接口来接受货物订购的请求，构件内部在下了订单以后向外执行请求运输的操作，得到请求运货成功，订单则被接受，否则订单取消。构件 Ship 的 ship_request 接口来接收运输的请求，内部功能请求运货成功，返回成功的请求，否则返回 fail。在这两个构件间建立和交互关系以后，对于构件系统的使用者来说，系统从功能上来讲是要实现订货。具体的过程表现为：下完订单再去查看是否能安排货物运送的服务，能及时运货的情况下才接受订单，否则要求退回订单，这也就是系统所体现出来的行为。

系统在运行一段时间后，为适应 Internet 的开放性和用户需求的变化，构件开始演化。例如构件 Order 的功能实现有改变，要求在订货后要增加检查客户的账户的功能，看客户是否已经超过可以赊账的上限，账户允许的条件下才接受订单。如果账户不容许，就要取消订单。针对这个需求，通过构件功能行为的修改来实现，图 1 中的(b)和(c)显示了两种可能的演化方式。在图 1(b)中，增加的账户检查功能和运输预定是并发执行的，在选择是接收还是取消订单前，这两个功能都要执行。从系统的行为来看，依然保持了进行运输预定通过的情况下才同意订单。而在图 1(c)中，演化后增加的账户检查功能和运输预定是选择执行的，这两个功能一次只会执行一个。对于使用者看到的系统行为，只要账户检查是成功的，那么订单就会被接受，运输预定则在检查账户的情况下不会执行。这样的情况显然是与系统原来的行为是不一致的，没有延续系统原来的功能。这样，系统功能执行出现错误，无法满足系统的要求，这就是说构件功能行为的改变有可能导致系统行为的错误。

3 构件系统行为模型

以上描述了基于构件的软件系统，在其构件的功能实现演化时，其系统行为的变迁可能存在的问题。其原因在于构件演化后，系统偏离了原来的行为，从而导致了错误，也就是违反了系统行为的一致性约束。如何知道怎样的构件演化会导致系统偏离原来的行为呢？我们发现基于构件的软件系统，构成系统行为的元素包括构件的功能行为和构件间的交互行为，因此在本节中我们首先在定义的系统实例的模型中详细描述构件的功能实现和构件间的交互的全部信息，再基于这些信息推导出系统行为的执行序列，即系统行为的描述。通过这种方式可以建立构件功能实现和系统行为之间的关联，构件功能实现的变化就能在系统行为的描述中反映出来。在下节中将介绍如何通过采用行为一致性约束条件，判断什

么样的构件演化是不符合约束条件的。这样，在系统描述模型中，随着构件功能实现的变化，系统行为的变化就会在系统行为执行序列中得到反映。

3.1 构件功能行为和构件间交互行为

下面我们将首先给出系统实现的模型描述。在这个概念模型中，包括组成系统的构件的功能实现和构件间的交互。构件的功能实现不仅包括描述构件生命周期的基于 WF-net[10] 的构件行为模型，还有构件中的实现方法、接口等信息。构件间的交互则是构件接口间的连接关系，并由此得到的 Petri-net 表示的交互行为描述。

定义 1（系统实例） 对于一个系统实例 S 的描述是一个三元组$\langle I_s, C_s, L_s\rangle$：

- I_s 是系统的名称标识。
- C_s 是构成系统的构件的集合。
- L_s 是构件间的交互关系的集合。

本小节的余下内容将给出构件以及构件间关系的具体定义。

定义 2（构件） 对于一个构件 C 的描述是这样一个元组$\langle I_c, M_c, P_c, MP, PM, BM_c\rangle$：

- I_c 是构件的名称标识。
- M_c 是构件功能实现的方法的集合。
- P_c 是构件提供的外部接口的集合。
- PM：$P_c \rightarrow M_c$，这个关联函数表明了构件提供的外部接口与构件内部的某个功能实现方法的关联关系，这表明被映射的内部功能函数会在外部表现出来，展现为系统的可观察行为。
- BM_c 是构件功能实现的行为模型，这个表示了构件 C 的生命周期。

紧接着的定义 3 是构件功能实现的行为模型的具体描述。

定义 3（构件功能实现的行为模型） 构件功能实现的行为模型的定义 BM_c 是基于 WF-net 的，$BM_c(P, T, F, i, o, TM)$ 是 WF-net 当且仅当满足下面的条件：

- P 是库所的有限集合，
- T 是变迁的有限集合，而且有 $P \cap T = \phi$，
- F 有向边的集合，且 $F \in (P \times T) \cup (T \times P)$，
- i 是 BM_c 的输入库所，即没有 $t \in T$，使得 $(t, i) \in F$，
- o 是 BM_c 的输出库所，即没有 $t \in T$，使得 $(o, t) \in F$，
- TM：$T_{BM} \rightarrow M_c \cup \{\varepsilon\}$，这是一个标记函数。即给在每个变迁定义了构件 C 中的功能实现方法，如果没有对应的功能实现的话则为 ε。如果 m 被连接到了 T 中的多个库所，将会给它添加一个唯一的下标，形式为 m_i。M_c 代表的是构件中的功能实现方法的集合。
- 连通性：加入一个不在 $P \cup T$ 中的新的标签 t'，新的 *Petri Net* $N' = (P, T \cup \{t\}, F \cup \{(o, t'), (t', i)\}, TM \cup \{(t', \tau)\})$ 是强连接的。

定义 4（构件间的交互） 构件间的交互关系是这样一个元组$\langle T_l, BM_l\rangle$，在这里要用到构件描述中关于构件接口的内容：

- T_l 用二元组的形式表示了构件接口间的连接关系，例如：$\langle m, n\rangle$，$m \in P_{ci}$，$n \in P_{cj}$，P_{ci} 和 P_{cj} 是系统中两个构件的接口的集合。
- BM_l 是通过二元组得到的用 Petri-net 表示的构件间交互关系的行为模型。这样一个行为模型的在具体内容在定义 5 中给出。

定义 5（构件间交互的行为模型） 构件交互的行为模型 BM 是在 Petri-net 模型的基础上加入了一个映射函数 TP。

- P 是库所的有限集合。
- T 是变迁的有限集合，而且有 $P \cap T = \phi$。
- F 有向边的集合，且 $F \in (P \times T) \cup (T \times P)$。
- TP：$T \rightarrow \{P_{c1}, P_{c2}, \cdots, P_{cn}\} \cup \{\varepsilon\}$，这是一个标记函数。$\{P_{c1}, P_{c2}, \cdots, P_{cn}\}$ 是系统中所有构件的外部接口的集合。这个函数给在每个变迁定义了组成系统的组件所提供的接口，如果没有对应的接口的话则为 ε。如果 p 被连接到了 T 中的多个库所，将会给它添加一个唯一的下标，形式为 p_i。P_c 代表的是组成系统的组件提供的接口的集合。因为在构件的描述中，构件的接口和构件的功能实现方法是关联在一起的，在这里通过构件的 PM 函数，这个模型转换得到描述构件的功能实现方法之间的同步关系，那么行为模型中变迁的标识就转换为了构件的功能实现方法。

3.2 系统行为

通过以上系统的描述，各个构件的功能行为可以通过构件间的交互统一在一起，形成一个统一的网络。下面通过给出系统状态和执行规则，可以导出整个系统的行为执行。下面将介绍如何通过以上信息推导得到系统行为的执行序列的集合。在这个集合的基础之上，我们参考 Aalst 在工作流挖掘中的工作，通过算法生成对应的 Petri-net 形式的系统行为图。

定义 6（系统状态） 系统状态是系统运行的基础，系统状态 r 表示的是系统描述中所有行为模型$\{BM_{c1}, BM_{c2}, \cdots, BM_{cn}, BM_{l1}, BM_{l2}, \cdots, BM_{lm}\}$中的标记的集合。初始的标记 M_0 是系统的所有构件功能实现的行为模型的输入库所都被标记，即$\{i_{c1}, i_{c2}, \cdots, i_{cn}\}$。

定义 7（变迁点火） 在系统状态定义的基础之上，我们给出了系统执行的条件及其过程。在系统状态 r 的条件下构件的功能实现模型中的变迁可以点火的条件包括：(1)在变迁所在的构件功能实现的行为模型中，它的输入库所都有一个标记。(2)如果这个变迁所关联的构件功能实现方法也标记了构件交互的行为模型中的一个变迁，那么那个变迁的输入库所也要有标记。而变迁的执行过程为：在这个变迁被执行完之后，所有在构件的功能实现和构件交互的行为模型中变迁 t 的输入库所的标记都要被移除，然后给在这两个模型中的变迁 t 的输出库所加入标记，得到系统的下一个状态 r'，这个可以被表示为 $r[t>r'$，r' 可以直接从 r 达到。

定义 8（状态序列） 这里将定义系统状态序列的合法性，因为只有从初始状态，通过变迁点火执行得到的系统状态序列才能体现系统状态变迁过程，即合法的。合法性的形式化定义是，如果系统状态 $r_1 = \{i_{c1}, i_{c2}, \cdots, i_{cn}\}$ 且 $\exists t$ 使得 $r_i[t>r_{i+1}$，则状态序列$\langle r_1, \cdots, r_n\rangle$是合法的。

例子 1 如图 1(a)中所示$\langle${Order. i, Ship. i}, {Order. $p1$, Ship. i}, {Order. $p2$, Ship. i}, {Order. $p2$, Ship. $p1$}, {Order. $p2$, Ship. $p2$}, {Order. $p2$, $s2$, Ship. o}, {Order. $p3$, Ship. o}, {Order. $p4$, Ship. o}, {Order. o, Ship. o}$\rangle$是系统 S_{Impl} 合法的状态序列。

定义 9（执行序列） 系统中执行序列$\langle t_1, \cdots, t_n\rangle$能从一个合法的系统状态序列$\langle r, \cdots, r_n\rangle$得到从 1 到 n 的所有执行 t_i，也就是满足 $r_i[t_i > r_{i+1}$，这个称这个序列时合法的。如果存在 $m_i = TM(t_i)$，则功能方法序列$\langle m_1, \cdots, m_n\rangle$也是合法的。

系统合法的执行序列体现的是系统的执行过程，系统所有合法的行为序列的集合 LMS_s 就是系统行为的动态运行结果的表示，有了这个集合也就确定了整个系统的行为，而下文中系统行为的一致性检查就是基于执行序列集合的。

例子 2 如图 1(a)中所示系统 $S_{lm\ pl1}$ 中所有合法的执行序列包括〈{Order}，{ Order，ship_request!}，{ Order，ship_request!，ship_request? }，{ Order，ship_request!，ship_request?，Ship_response! }，{Order，ship_request!，ship_request?，Ship_response!，Ship_respones?}，{ Order，ship_request!，ship_request?，Ship_response!，Ship_respones?，Accept_order }，{ Order，ship_request!，ship_request?，Ship_response!，Ship_respones?，Accept_order，Log }，{ Order，ship_request!，ship_request?，Ship_response!，Ship_respones?，Cancle_order }〉，{ Order，ship_request!，ship_request?，Ship_response!，Ship_respones?，Cancle_order，Log }〉。

为了更直观地表示，我们参考了 Aalst 在工作流挖掘中的工作。在文[15]中，作者通过算法利用系统的工作日志得到了其模型的表示。其过程为通过筛选日志内容，得到工作流的执行序列，并从序列得到工作流的模式。借鉴其工作，我们通过算法将系统行为执行序列中的集合转化为系统行为的 Petri-net 表示。图 2 中的(a)，(b)，(c)分别对应图 1 中(a)，(b)，(c)所示的系统的行为图形表示。

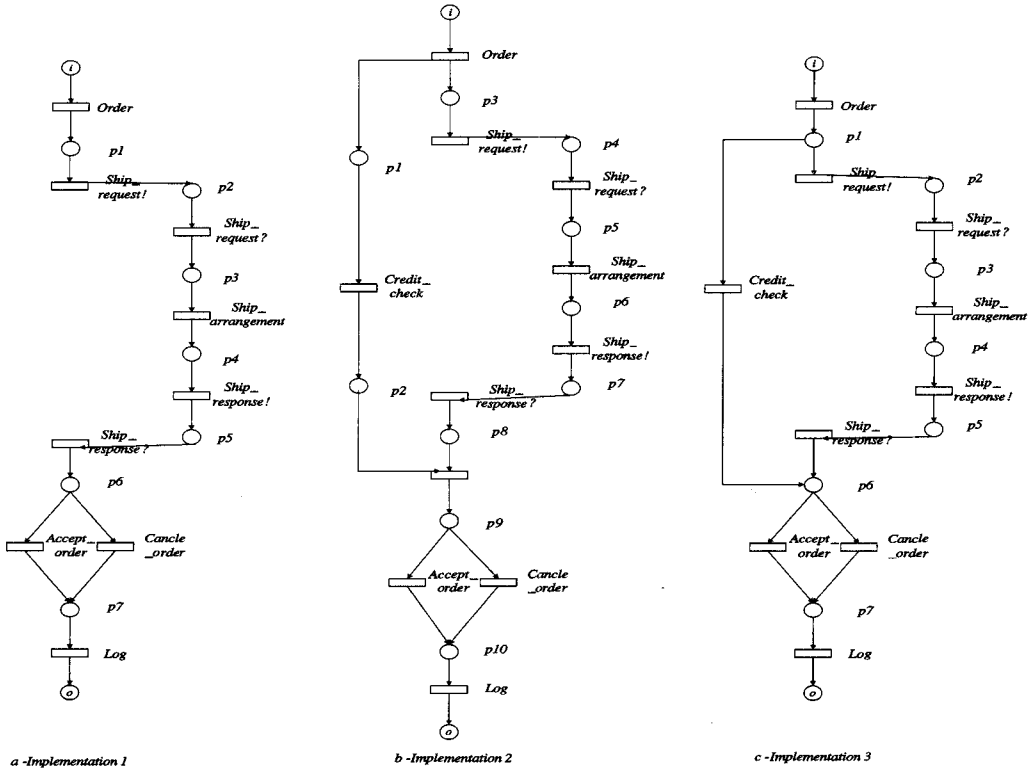

图 2

4 构件演化时的系统行为的一致性约束

通过上面的工作，可以从系统描述中包括的构件功能实现和构件间的交互推导得到系统行为描述，由此建立了构件的功能实现和系统行为之间的关联关系。下面将介绍在构件演化时，如何对系统行为进行一致性检查，避免不合法的构件演化。

4.1 一致性约束规则

在本文中使用的一致性规则称之为观察行为一致性，这个定义借鉴了 Ebert 和 Engles 为定义类及其子类行为描述的相互关系而提出的对象行为和相应的一致性关系：观察一致性[13]。所谓观察一致性是指把系统目标行为看成可观察操作序列的描述，若忽略实际系统行为实现中不能被观察的操作，系统实现行为中的可观察方法序列在原有系统行为中也可被观察。在这里，不能被观察的定义取决于实际的语意环境，比如说内部操作一般被认为是不可观察的。因为系统所有合法的行为序列就是系统行为的动态运行结果的体现，因此在这个规则的约束下，能保证新系统行为中执行的功能序列是继承自原来的系统，保证系统行为不会偏离原来的系统，避免系统错误的产生。行为观察一致性规则在上文定义的系统行为的描述中表示如下：

定义 10(一致性规则) 系统 S' 和另一个系统 S 是行为一致的当且仅当 $(LMS_s' \uparrow M_s') \in (LMS_s \uparrow M_s)$。即对于属于系统 S' 行为执行序列集合 LMS_s' 的每一个操作序 lms_s'，$(lms_s' \uparrow M_s') \in (LMS_s \uparrow M_s)$，这里的 $\langle m_1, \cdots, m_p \rangle \uparrow M_s' = \langle m_1, \cdots, m_q \rangle$ 满足所有 $\langle m_1, \cdots, m_q \rangle$ 中的 m_i 都属于 M_s'。M_s 根据系统语意给出的在行为模型中可观察的操作和集合。

例子 3 在上面的例子中，系统中可观察的操作是与接口连接的可被外部环境调用的操作，构件在演化时其外部接口没有发生变化，其对应的功能实现也还是一样的。因此 M ={ order，ship_request!，ship_request?，Ship_response!，Ship_respones?，Cancle_order，Accept_order}。系统 $S_{lm\ pl2}$ 和系统 $S_{lm\ pl1}$ 的行为是满足行为一致性规则的，因为 $(LMS_{lm\ pl2} \uparrow M_{lm\ pl2}) \in (LMS_{lm\ pl1} \uparrow M_{lm\ pl1})$。而系统 $S_{lm\ pl3}$ 的行为则是与 $S_{lm\ pl1}$ 不满足行为一致性规则的。

4.2 构件演化时的系统行为一致性约束检查

在软件的集成开发环境中，如果要对构件进行演化，首先在系统实例的描述中改变构件的功能实现的描述，同时推导生成新的系统行为执行序列集合，也就可以得到系统的行为的变化。通过上文定义一致性约束规则检查，可知构件按此演化后系统行为是否满足的行为一致性约束条件。如果系统新的行为不符合一致性规则的约束，就认为系统新的行为不是行为一致的。那么，这样的构件演化则不被实施。

5 系统实现

构件演化的策略是由支撑平台实施的。如果要在构件演化时对系统行为进行一致性的检查约束，需要在支撑平台加入系统实例实现的描述信息，并且给出行为一致性规则在系统行为上的约束检查，这样，构件演化时系统行为就会被监控和约束来满足这些规则的要求。基于以上的研究工作，我们设计并开发了 SOBECA(Service-Oriented Behavior and Capability support Architecture)，该系统包括一个构建在 OSGi[16] 平台上的运行支撑环境和一个以 Eclipse Plugin 形式提供的集成开发工具，因此该平台能够在构件的集成和动态演化时支持行为的一致性检查。图 3 是 SOBECA 系统的基础架构。

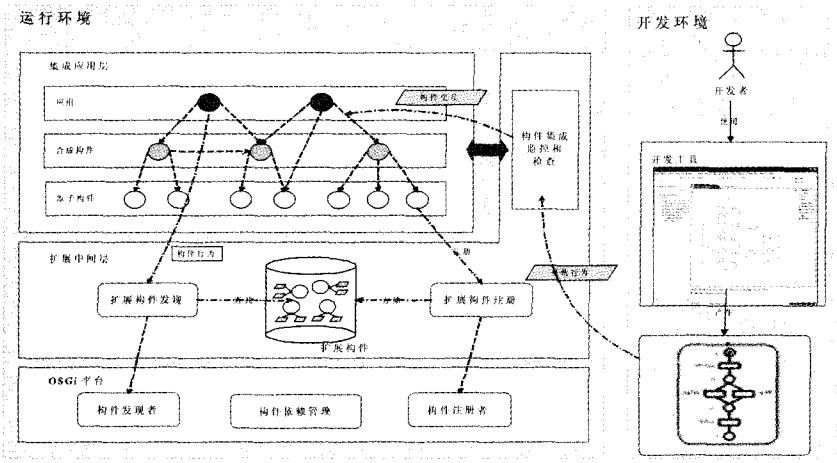

图 3

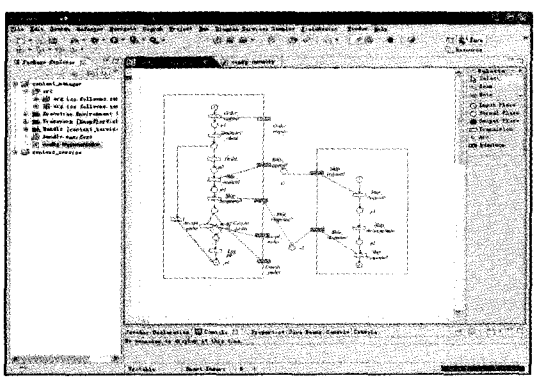

图 4

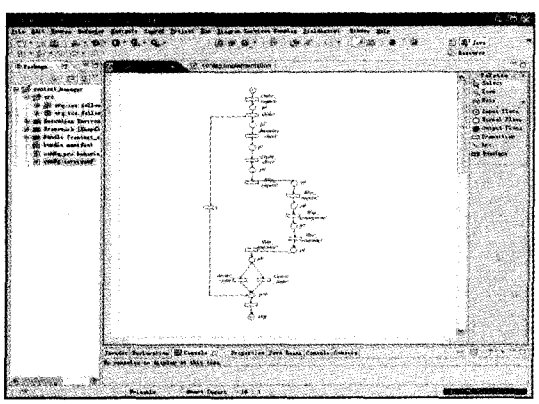

图 5

SOBECA 系统的运行环境自底向上分为 OSGi 平台、扩展中间件和集成应用层三部分。OSGi 平台提供了一些用于部署可扩展和加载的组件的设施，这些组件通过接口对外提供服务；扩展中间层扩展了原来对组件发现、注册和替换机制，加入了构件行为，并在运行时检测构件行为，这个部分的内容已经在我们以前的工作[17]中介绍过；集成应用层则通过图形子模块来描述系统的组成，即构件以及构件间交互等信息，基于扩展中间层及以上信息能推导得到系统的功能行为执行序列。其次，内嵌的行为一致性检查子模块能够按照行为一致性规则的要求对定义的系统功能行为进行验证，以保证其符合系统行为的一致性规则约束。

SOBECA 的开发环境是基于 Eclipse 的采用 Eclipse Plugin 的形式。图 4、5 是开发工具提供典型的图形用户界面，图 4 表示系统实现，图 5 表示生成的系统行为。开发人员可以通过拖放图形元素来定义构件行为和交互，提供了一种直观的构件集成描述手段；一致性验证机制可以在开发及演化时检查系统功能行为是不是满足一致性的要求，在某种程度上保障了构件演化的有效性，以此来保证实施的构件演化都是有效的。

相关工作和总结 在本文中我们主要分析构件演化的条件下，如何来保证系统行为的一致性。在以往的工作中，例如文[8]和[12]中，通过在新构件和原来的构件间建立相容性和替换性，基于局部构件行为的一致性来实现全局的系统行为的一致性，来保证原来系统环境下对新构件使用不会导致系统行为出错。这样的做法，一致性要求较高，降低了演化的灵活度，不能满足不同系统、不同环境下对构件演化的需求。而在本文中通过系统行为层面而不是在构件行为层面加入一致性约束，来保证系统的功能行为的延续，在新的系统的功能行为是延续了原来系统功能行为的情况下，系统行为执行的正

(下转第 300 页)

对于未标记的状态对(A,E)、(A,G)、(B,H)、(D,F)、(E,G)采用如上的方法分别同时进行考察,执行的结果如表5所示。

表5 实例执行第三步得到的可区分状态表结构

B	×	-	-	-	-	-	-
C	×	×	-	-	-	-	-
D	×	×	×	-	-	-	-
E		×	×	×	-	-	-
F	×	×	×		×	-	-
G	×	×	×	×	×	×	-
H	×		×	×	×	×	×
	A	B	C	D	E	F	G

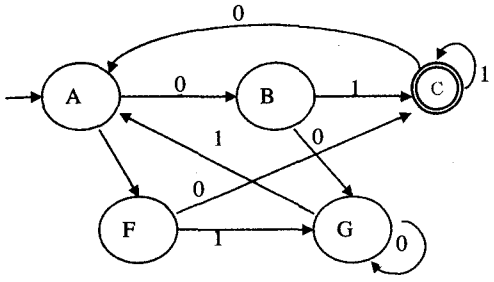

图2 最小化的DAF

对剩余的未标记的状态对(A,E)、(B,H)、(D,F)也执行上述操作,得到结果和表5相同,可区分状态表不再发生变化,终止循环,从而得到最终的可区分状态表,同表5。

(3)从表5中找出未标记的状态对(A,E)、(B,H)、(D,F),这三对状态分别是等价的,从而可以得到最小化的DFA,如图2所示。

结束语 确定有限自动机的最小化并行算法的设计及过程分析有其理论和实践意义,该算法利用可区分与不可区分状态的特点实现了在并行环境中的有限自动机的最小化。但算法的性能以及并行结构模型下的存储结构,数据分布等问题还需要进一步的研究。

参考文献

1 Hopcoft J E, Ullman J D. Introduction to Automata Theory, Languages and Computation (Second Edition). Beijing : China Machine Press, 2004

2 陈火旺,刘春林,谭庆平,等. 程序设计编译原理. 北京:国防工业出版社,2003

3 蒋宗礼,姜守旭. 形式语言与自动机. 北京:清华大学出版社,2002

4 Grandi P. Implementing (nondeterministic) parallel assignments. Information Processing Letters , 1996, 58: 177～179

5 Heng A C K, Low M Y H. Loop parallelization tool for message-passing systems. Microprocessors and Microsystems , 1997, 20: 409～421

6 Gibbons A, Rytter W. Efficient parallel parsing algorithms. Cambridge University Press, 1990

7 陈国良. 并行算法的设计与分析. 北京:高等教育出版社,2002

(上接第270页)

确性就可以得到保证。这样,不需要对替换的构件行为进行严格的行为约束,又保证了运行时的系统行为不会因此偏离原来的系统而产生错误。因此可以在演化的过程中,保证系统的行为保持一致,又保持构件演化的灵活行。

在本文中使用的一致性理论是行为的观察一致性理论,其理论研究在不同的论文中有用状态变迁图[13]、进程代数[14]、Petri-net[14]来描述对象行为和行为间的继承。我们在将来的研究中,希望对一致性规则的约束进行进一步研究,讨论在不同的应用场景下适合的和灵活更大的一致性约束规则及其应用。

参考文献

1 Yang F Q, Mei H, Li K Q. Software reuse and software component technology. Acta Electronica Sinca, 1999,27(2):68～75 (in Chinese with English abstract)

2 Szyperski C. Component software: Beyond object-oriented programming(first edition). New York: Addison-wesley, 1997

3 Hememnan G T, Ohlenbuech H M. An Evaluation of Component Adaptation Techniques: [Technical Report]. WPICS-TR-99-20. Department of Computer Science, Worcester Poly technical Institute, Feb 1999

4 Sun J. Dong J S. Design synthesis from interaction and state-based specifications. In: Transactions on Software Engineering, IEEE, 2006

5 Magee J, Kramer J. Dynamic structure in software architectures. In: Kaise G E. ed. Proceedings of the ACM SIGSOFT'96: the 4th Symposium on the Foundations of Software Engineering. New York: ACM Press, 1996. 3～14

6 Plasil F, Visnovsky S. Behavior Protocols for Software Components. In: Transactions on Software Engineering, IEEE, 2002

7 Allen R, Garlan D. Formalizing architectural connection. In: Proc CSE'94, Sorrento(Italy), 1994. 71～80

8 Canal C, Pimentel E, Troya J M. Compatibility and inheritance in software architectures. Science of Computer Programming, 2001,41(2):105～138

9 Nierstrasz O. Regular types for Active Objects. In: ACM SIGPLAN Notices, 28 (10); Proceedings of OOPSLA'93, Washington DC, 1993. 1～15

10 van der Aalst W M P. Verification of Workflow Nets. In: Az' ema P, Balbo G, editors. Application and Theory of Petri Nets 1997, vol 1248 of Lecture Notes in Computer Science. Berlin: Springer-Verlag, 1997. 407～426

11 Van der Aalst W M P, van Hee K M, van der Toorn R A. Component-based Software Architectures: A Framework Based on Inheritance of Behaviour. Working Paper Series 45, Eindhoven University of Technology, 2000

12 Hameurlain N. On compatibility and behavioural substitutability of component protocols. In: Software Engineering and Formal Methods, 2005(SEFM 2005). Third IEEE International Conference on Sept 2005. Digital Object Identifier 10. 1109/SEFM. 2005. 30. 2005. 394～403

13 Ebert J, Engels G. Specialization of Object Life Cycle Definition: [Technical Report]. Koblenz University, 1995. 19～95

14 Basten T, van der Aalst W M P. Inheritance of behavior. Journal of Logic and Algebraic Programming, 2001, 47(2): 47～145

15 van der Aalst W M P, Weijters A J M M, Maruster L. Workflow Mining: Discovering Process Models from Event Logs (Accepted for publication in IEEE Transactions on Knowledge and Data Engineering)

16 The Open Services Gateway Initiative (OSGi). www. osgi. org

17 殷琴,李俊,罗毅,等. 支持服务行为和质量的面向服务软件开发. 计算机科学(已录用)