组合服务控制流测试

余 莹 金茂忠 黄 宁

(北京航空航天大学 计算机学院, 北京 100191) (北京航空航天大学 工程系统工程系, 北京 100191)

摘 要: 结合Web服务本体语言(OWL-S, Web Ontology Language for Services)和线性时态逻辑理论(LTL, Linear Temporal Logic), 研究用于测试的组合服务流程形式化描述方法和动态测试信息分析方法. 将OWL-S作为组合服务的需求参考模型, 采用组合服务标准和形式化描述方法相结合的方式, 用线性时态逻辑刻画OWL-S控制结构的动态语义, 明确地表示出控制结构中各成分的执行顺序. 进一步用线性时态逻辑公式集合描述组合服务的控制流需求, 从而使原子服务的交互模式有了明确的表示. 基于这种交互模式表示, 采用LTL在有限状态序列上的语义, 对组合服务实现执行过程中获取的动态信息进行分析, 测试组合服务实现的执行过程与组合服务控制流需求的一致性.

关 键 词: 软件测试; 组合服务; 控制流; Web服务本体语言; 线性时态逻辑

中图分类号: TP 311.53

文献标识码: A **文章编号**: 1001-5965(2009)01-0117-05

Testing control flow of composite service

Yu Ying Jin Maozhong

(School of Computer Science and Technology, Beijing University of Aeronautics and Astronautics, Beijing 100191, China)

Huang Ning

(Dept. of System Engineering of Engineering Technology, Beijing University of Aeronautics and Astronautics, Beijing 100191, China)

Abstract: OWL-S (Web ontology language for services) and linear temporal logic were used to investigate formal method of describing composite service flow for testing and analyzing dynamic testing information. OWL-S was used as requirements reference model of composite service. Combination of composite service standard and formal method was adopted. Linear temporal logic was used to describe dynamic semantics of control constructs in OWL-S. Accordingly control flow requirements of composite service was described by a set of linear temporal logic formulae. Besides the existing advantage of OWL-S, the describing method can explicitly express atomic services interaction pattern and the describing result can be used as reference model of testing composite service implementation. Based on the explicit expression and semantic model of linear temporal logic on finite execution traces, dynamic testing information captured during composiste service execution was analyzed to test the correspondence between composite service execution and composite service control flow requirements.

Key words: software testing; composite service; control flow; OWL-S(Web ontology language for services); linear temporal logic

随着网络的发展, Web服务在电子商务和企业应用集成中得到了广泛的应用. 由于单个服务所能提供的功能有限, 通常会把多个原子服务组合起来, 构成组合服务, 从而实现更大粒度的功能. 通过组合多个原子服务而得到的组合服务实现是否满足组合服务的控制流需求, 是组合服务

收稿日期: 2008-03-11

基金项目: 国家十一五国防预研资助项目

作者简介: 余 莹(1976-), 女, 吉林长岭人, 博士生, fennyyu@126.com.

测试所要关注的一个重要方面. 关注组合服务内部的测试,可以分为关注模型和关注实现两类. 关注模型的研究,利用 Petri网等形式化方法的分析技术[1]或者模型检测[2],分析组合服务模型是否满足或违背需求特性,对组合服务模型进行验证. 关注实现的研究,从运行时监控的角度来研究组合服务实现,通过动态分析,判断组合服务实现的执行是否满足需求特性[3]. 本文以 OWL-S(Web Ontology Language for Services)[4]作为组合服务的需求模型,测试组合服务实现是否符合组合服务的控制流需求.

1 组合服务控制流的 LTL 描述

OWL-S是基于本体描述语言 OWL 定义的语义 Web服务描述语言,在 ServiceModel中定义了8种控制结构,表示组合服务与原子服务不同的交互模式. 这些控制结构基于本体语言进行定义,具有静态语义,但由于缺少明确的动态语义,控制结构中各成分的执行顺序没有显示的定义. 相应地,原子服务在组合服务控制流下的执行顺序也没有显示的定义. 这在进行组合服务控制流测试时,不易对测试中收集到的动态信息进行分析,以确定组合服务实现是否满足组合服务的控制流需求.

线性时态逻辑(LTL, Linear Temporal Logic)在命题逻辑的基础上引入时态算子,在线性时间上刻画事件的时态顺序,适合描述原子服务在组合服务控制流下的执行顺序. 测试在有限的执行序列上进行,文献[5]给出了 LTL在有限状态序列上的语义解释,本文将该解释作为 LTL的语义模型,用 LTL刻画 OWL-S控制结构的动态语义,进一步地,将组合服务的控制流用 LTL公式集合描述.

1.1 原子命题定义

为了用 LTL公式表达程序执行的特性,需要将所关注的程序执行事件用相应的原子命题定义. 设 P为原子命题集合, $\sigma = s_1 s_2 \cdots s_n$ 为一有限状态序列,每一个状态对应着一个在该状态中成立的原子命题集合 P_T, $P_T \subset P$. 设 p 为原子命题, $\forall p \in P$, $s_i(p) = \text{true}$表示原子命题 p 在状态 s_i 中成立. 4类原子命题的定义如下.

1) 交互类原子命题. 用原子服务名称表示这类原子命题,如果组合服务与某原子服务的交互事件发生,则相应的原子命题为 true,否则为 false.

2) 控制条件判断类原子命题. C 为控制条件在组合服务需求模型中的名称,原子命题 C_T 为 true表示控制条件判断为真的事件发生;原子命题 C_F为 true表示控制条件判断为假的事件发生.

3) 控制结构执行类原子命题. 用控制结构名称表示这类原子命题,如果需要执行某个控制结构,则相应的原子命题为 true,否则为 false.

4) 原子命题 end 为表示状态序列的结束,引入原子命题 end, end不与程序执行事件相对应. 当 end成立时,表明其后不再有执行事件产生. end的定义如下.

定义 1 原子命题 end

1) $\text{end} \in P$;

2) $\forall i$, $1 \leqslant i < n$, $s_i(\text{end}) = \text{false}$;

3) $\forall p \in P$, $p = \text{end} \rightarrow s_n(p) = \text{true}$并且 $p \neq \text{end} \rightarrow s_n(p) = \text{false}$.

1.2 控制结构的 LTL描述

OWL-S的 ServiceModel中包含 8种控制结构: Sequence, If-Then-Else, Repeat-While, Repeat-Until, Choice, AnyOrder, Split和 Split-Join,下面用 LTL分别描述这 8种控制结构的动态语义. 其中 C 代表控制条件; A, A_1, A_2, …, A_n代表原子服务或控制结构; end的含义同上节的定义.

Sequence表示该控制结构中的各成分按照顺序依次被执行,因此各成分之间是顺序的关系. Sequence($A_1, A_2, \cdots, A_n$)的动态语义用 LTL可以表示为公式(1). 其中, o代表 LTL中的“next”操作符, o^2为 o o的简写, o^3为 o o o的简写,依此类推,下同.

$$A_1 \wedge oA_2 \wedge \cdots \wedge o^{n-1}A_n \wedge o^n \text{ end} \quad (1)$$

为了将时态逻辑规约用于测试,文献[5]给出了在有穷状态序列上为 LTL公式生成测试预言的算法,能够判别一个有限状态序列是否满足 LTL公式. 本文将该算法简称为 Dillon算法,应用该算法给出用 LTL刻画 OWL-S控制结构的动态语义的证明.

证明 应用 Dilbn算法,由公式(1)生成的有限状态自动机如图 1a所示. 该自动机表示在顺序地接受 $A_1, A_2, \cdots, A_n$, end输入序列后,可以接受任意的输入. 根据 end的定义,当 end成立时,其后不再有测试事件产生,所以 end之后不会再有其它输入. 图 1a中的自动机可以简化为图 1b所示的自动机,该自动机表明在依次输入 $A_1, A_2, \cdots, A_n$, end后,到达终态,并且不再接受其它输入,能够表明 Sequence($A_1, A_2, \cdots, A_n$)的含义. 因此公式(1)能够表示 Sequence($A_1, A_2, \cdots, A_n$)的动态语义. 证毕

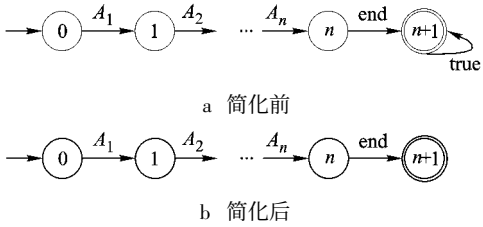


图 1　由公式(1)生成的有限状态自动机

以下公式的证明与此类似,所以省略了具体的证明过程.

If-Then-Else控制结构表示分支选择,两个分支中的哪个分支被执行,取决于对控制条件的判断.如果把对控制条件的判断和分支的执行作为两个独立的事件,那么它们的执行是顺序关系. If (C) Then A_1 Else A_2 的动态语义用 LTL 可以表示为公式(2).

$$(C_T \wedge o A_1 \wedge o^2 \text{ end}) \vee (C_F \wedge o A_2 \wedge o^2 \text{ end}) \quad (2)$$

Repeat-While控制结构表示 While循环,当条件成立时,循环体被执行;当条件不成立时,则退出循环. Repeat(A)While(C)的动态语义用 LTL 可以表示为公式(3). 其中"U"表示 LTL中的"Until"操作符.

$$(C_F \wedge o \text{ end}) \vee (C_T \wedge (((C_T \wedge oA) \vee (A \wedge oC_T)) U (A \wedge oC_F \wedge o^2 \text{ end}))) \quad (3)$$

Repeat-Until控制结构表示 Until循环,循环体被执行,直到条件成立时退出循环. 与 Repeat-While的区别在于循环体至少被执行一次. Repeat(A)Until(C)的动态语义用 LTL可以表示为公式(4).

$$A \wedge (((A \wedge oC_F) \vee (C_F \wedge oA)) U (A \wedge oC_T \wedge o^2 \text{ end})) \quad (4)$$

Choice控制结构表示从多个分支中任选一个执行, Choice$(A_1, A_2, \cdots, A_n)$的动态语义用 LTL可以表示为公式(5).

$$(A_1 \wedge o \text{ end}) \vee (A_2 \wedge o \text{ end}) \vee \cdots \vee (A_n \wedge o \text{ end}) \quad (5)$$

AnyOrder控制结构表示多个分支以任意的顺序执行,所以这多个分支的顺序的全排列结果就是所有可能的执行情况.由于一旦执行,执行顺序就是确定的,具有唯一性,所以应该是全排列结果的合取关系. AnyOrder$(A_1, A_2, \cdots, A_n)$的动态语义用 LTL可以表示为公式(6).

$$(A_1 \wedge oA_2 \wedge \cdots \wedge o^{n-1} A_n \wedge o^n \text{ end}) \vee (A_1 \wedge oA_3 \wedge \cdots \wedge o^{n-1} A_n \wedge o^n \text{ end}) \vee \cdots \vee (A_n \wedge oA_{n-1} \wedge \cdots \wedge o^{n-1} A_1 \wedge o^n \text{ end}) \quad (6)$$

Split控制结构表示内部的各个分支可以并发执行,不需要同步,只要各个分支准备好执行,Split控制结构就可以准备结束,即各个分支的开始事件和 Split控制结构的结束事件可以以任意的、不连贯的顺序执行.各个分支内部的执行顺序在各分支内部保持,不需要考虑各个分支之间的顺序关系.

定义 **2**　IncoherentAnyOrder$(A_1, A_2, \cdots, A_n)$表示 $A_1, A_2, \cdots A_n$ 可以以任意的、不连贯的顺序执行.

定义 2用 LTL可以表示为公式(7). 其中"<>"表示 LTL中的"eventually"操作符.

$$(A_1 \wedge o<> (A_2 \wedge \cdots o<> (A_n \wedge o<> \text{end}))) \vee (A_1 \wedge o<> (A_3 \wedge \cdots o<> (A_n \wedge o<> \text{end}))) \vee \cdots \vee (A_n \wedge o<> (A_{n-1} \wedge \cdots o<> (A_1 \wedge o<> \text{end}))) \quad (7)$$

定义 **3**　函数 se(A)表示 A 的第 1个事件.如果 A 是原子过程,则 se(A)就表示组合服务和该原子服务的交互事件;如果 A 是控制结构,则 se(A)表示执行该控制结构的第 1个事件.

Split$(A_1, A_2, \cdots, A_n)$的动态语义用 LTL可以表示为公式(8),可以根据公式(7)进一步展开.

$$\text{IncoherentAnyOrder}(\text{se}(A_1), \text{se}(A_2), \cdots, \text{se}(A_n)) \quad (8)$$

Split-Join控制结构表示内部的各个分支可以并发执行,需要同步,同步方式为障碍同步,即所有的分支都执行完成后, Split-Join控制结构才执行完成.各个分支内部的执行顺序在各分支内部保持,不需要考虑各个分支之间的顺序关系.

定义 **4**　$\forall i, 1 \leqslant i \leqslant n$, A_i 内发生的事件集合为 $\{e_{i1}, e_{i2}, \cdots, e_{ij_i}\}$. 如果 A_i 为原子过程,则集合中至多含一个元素;如果 A_i 为控制结构,则集合中可能含有多个元素.

定义 **5**　$\forall i, 1 \leqslant i \leqslant n$, $de_i = e_{i1} \vee e_{i2} \vee \cdots \vee e_{ij_i}$, $de = de_1 \vee de_2 \vee \cdots \vee de_n$,将其展开得到 $(e_{11} \vee e_{12} \vee \cdots \vee e_{1j_1}) \vee (e_{21} \vee e_{22} \vee \cdots \vee e_{2j_2}) \vee \cdots \vee (e_{n1} \vee e_{n2} \vee \cdots \vee e_{nj_n})$. de表示 Split-Join控制结构内所有事件的合取关系.

根据 Split-Join 控制结构的含义, Split-Join $(A_1, A_2, \cdots, A_n)$的动态语义用 LTL可以表示为公式(9),可以根据公式(7)进一步展开.

$$\text{IncoherentAnyOrder}(\text{se}(A_1), \cdots, \text{se}(A_n)) \wedge (de \, U \, \text{end}) \quad (9)$$

1.3　控制流的 LTL描述

从执行的角度看,控制流是一个流图,表现为图结构;从结构层次的角度看,控制流是一个有序

树结构, 根节点和中间节点是控制结构, 叶子节点是原子服务. 用 LTL描述控制流, 从结构层次的角度进行. 对于任一层次的非叶子节点, 它的控制结构类型决定了它的后继节点如何被执行, 这一执行的动态语义即可用控制结构的 LTL描述来表达. 控制流的 LTL描述算法如下所示.

算法: 控制流的 LTL描述.

输入: OWL-S模型, 控制结构的 LTL描述.

输出: LTL公式集合.

1) 按照层次遍历算法遍历 OWL-S模型中组合服务控制流的有序树;

2) 如果当前结点为控制结构, 取得该结点的所有后继结点, 根据控制结构的类型, 按照相应的 LTL描述规则进行描述, 输出当前结点名称、对应的 LTL公式和所属结构层次.

2 组合服务控制流测试

对组合服务实现进行测试时, 从执行过程中获取的动态信息表征了原子服务的实际交互过程. 将 LTL描述的交互模式用于对动态信息的分析, 可以基于交互模式匹配的分析方法, 确定组合服务实现的实际执行过程是否满足组合服务的控制流需求.

2.1 动态信息序列

进行组合服务的控制流测试, 主要关注以下2类动态信息, 它们构成了表达组合服务实际执行过程的动态信息序列:

1) 交互事件. 组合服务与原子服务的交互事件, 即组合服务调用原子服务的事件, 事件名称用原子服务的名称来表示.

2) 控制条件判断事件. 组合服务需求模型中, 如果存在控制条件, 则记录在组合服务执行过程中对控制条件的判断. C 为控制条件在组合服务需求模型中的名称, C_T为 true表示控制条件判断为真的事件发生, C_F为 true表示控制条件判断为假的事件发生.

2.2 动态信息分析

如果在一个执行事件序列片断的尾部加上“end”后, 满足一个 LTL公式, 并且去掉其中的任意元素之后, 都不再满足, 则称该执行事件序列片断匹配 LTL公式所表达的交互模式. 对执行事件序列进行分析, 从中识别出这样的一个一个片断, 并用相应的控制结构名称来替换执行事件序列片断, 采用自底向上的方式进行聚合, 直至聚合到控制流有序树的最顶层, 即认为执行事件序列符合控制流; 否则, 执行事件序列违背控制流, 组合服务实现中存在错误. 执行事件序列的分析算法如下所示.

算法: 执行事件序列分析.

输入: 执行事件序列, 控制流 LTL公式集合.

输出: 组合服务的执行过程是否满足控制流需求.

1) 从最大层次的 LTL公式开始自底向上进行分析;

2) 如果执行事件序列中的某一子序列在追加了 end元素后, 满足当前公式, 则用当前公式对应的控制结构名称替换该子序列;

3) 继续分析, 直到没有公式可满足.

4) 如果最后的执行事件序列与最顶层公式对应的控制结构名称一致, 则组合服务实现的执行过程满足组合服务的控制流需求; 否则不满足.

3 系统实现

3.1 系统体系结构

依据上述研究方法, 实现了一个组合服务控制流测试的原型系统, 系统体系结构如图 2所示.

1) 控制结构的 LTL描述规则. 用 LTL刻画 OWL-S的控制结构的动态语义, 将控制结构用 LTL描述.

2) 组合服务控制流转换. 以 OWL-S描述组合服务的控制流需求, 应用控制流的 LTL描述算法, 将组合服务控制流需求描述为 LTL公式集合.

3) 组合服务执行事件序列分析. 基于 LTL公式集合描述的组合服务控制流需求, 对测试过程中收集到的动态信息进行分析, 应用执行事件序列分析算法, 判断组合服务的执行过程是否符合组合服务的控制流需求. 其中执行序列片断是否满足 LTL公式的解释器采用文献[6]的方案.

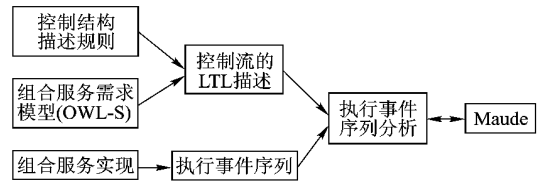


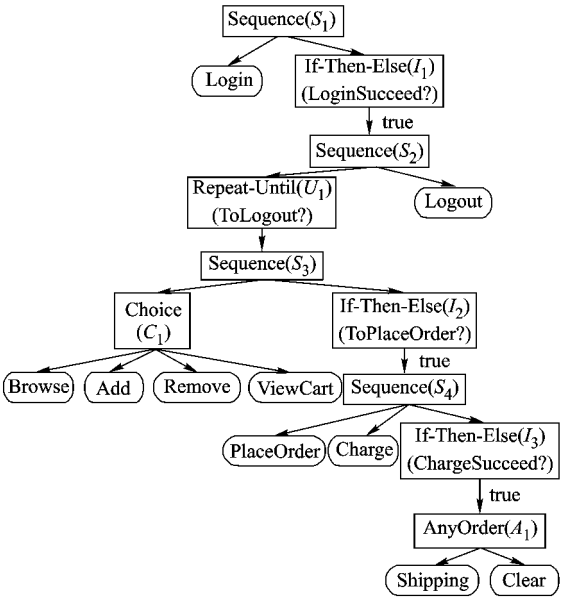
图 2 系统体系结构

3.2 实 例

本节以一个 OnlineShopping组合服务为例, 说明本文提出的组合服务控制流测试方法. 控制流需求的有序树结构表示如图 3所示, 将图 3的有序树结构用 LTL描述, 如表 1所示.

为验证测试方法, 用 Java语言实现了该组合

服务. 对该组合服务实现进行测试, 它的一次执行获得如下事件序列: Login, LoginSucceed$_T$, Browse, ToPlaceOrder$_F$, ToLogout$_F$, Add, ToPlaceOrder$_F$, PlaceOrder, Charge, ChargeSucceed$_T$, Clear, Shipping, ToLogout$_T$, Logout. 通过分析, Clear, Shipping这个子序列在追加了 end元素后, 满足控制结构 A_1对应的 LTL公式, 因此用 A_1替换该子序列, 得到聚合后的事件序列 Login, LoginSucceed$_T$, Browse, ToPlaceOrder$_F$, ToLogout$_F$, Add, ToPlaceOrder$_T$, PlaceOrder, Charge, ChargeSucceed$_T$, A_1, ToLogout$_T$, Logout



圆角矩形节点代表原子服务, 节点内容为原子服务名称; 矩形节点代表控制结构, 节点内容为控制结构类型和控制结构名称, 对于含条件判断的控制结构, 节点内容中还包含控制条件的表示.

图 3　OnlineShopping组合服务控制流的有序树结构

表 1　用 **LTL**描述 **OnlineShopping**组合服务控制流

控制结构名称	LTL公式	所属结构层次
S_1	Login∧ oI_1∧ o o end	1
I_1	(LoginSucceed$_T$∧ oS_2∧ o o end)∨ (LoginSucceed$_F$∧ o end)	2
S_2	U_1∧ oLogout∧ o o end	3
U_1	S_3∧ (((S$_3$∧ o ToLogout$_F$)∨ (ToLogout$_F$∧ oS_3)))U(S$_3$∧ oToLogout$_T$∧ o o end)	4
S_3	C_1∧ oI_2∧ o o end	5
C_1	(Browse∧ o end)∨ (Add∧ o end)∨ (Remove∧ o end)∨ (ViewCart∧ o end)	6
I_2	(ToPlaceOrder$_T$∧ oS_4∧ o o end)∨ (ToPlaceOrder$_F$∧ o end)	6
S_4	PlaceOrder∧ o Charge∧ o o I_3∧ o o o end	7
I_3	(ChargeSucceed$_T$∧ oA_1∧ o o end)∨ (ChargeSucceed$_F$∧ o end)	8
A_1	(Shipping∧ oClear∧ o o end)∨ (Clear∧ o Shipping∧ o o end)	9

依次进行分析, 最后聚合得到 Login, LoginSucceed$_T$, S_3, ToLogout$_T$, S_3, S_4, ToLogout$_T$, Logout事件序列. 若组合服务实现正确, 应获得"S_1"这样的分析结果, 因此组合服务实现中存在错误, 不符合组合服务的控制流需求. 事实上, 当出现 ToPlaceOrder$_T$事件时, 其后不应出现 PlaceOrder事件, 可见组合服务实现中确实存在错误.

4　结束语

本文结合组合服务标准 OWL-S和线性时态逻辑理论, 研究用于测试的组合服务流程形式化描述方法和动态测试信息分析方法. 以便于测试分析为描述目的, 对组合服务流程进行形式化描述, 采用组合服务标准和形式化描述方法相结合的方式, 前端以得到广泛应用的组合服务标准 OWL-S作为组合服务的需求模型, 后端以 LTL形式化地描述组合服务流程. 既为组合服务与原子服务的交互模式提供明确、显式的定义, 便于测试分析; 又使用户不必关心和掌握形式化方法的细节, 只需专注在前端建模上. 在动态测试信息分析方面, 采用 LTL在有限状态序列上的语义, 将 LTL描述的清晰、无二义性的组合服务流程, 用于动态测试信息的自动分析. 进一步工作可结合组合服务控制流测试中所需要的动态信息, 对组合服务实现执行过程中的动态信息获取方法展开研究.

参考文献 (**References**)

[1] Narayanan S, McIlraith S. Simulation, verification and automated composition of web services[C]//Lassner D, Roure D D, Iyengar A. International World Wide Web Conference. Honolulu: ACM, 2002: 77-88

[2] Fu X, Bultan T, Su J. Analysis of interacting BPEL web services[C]//Feldman S, Uretsky M, Najork M, et al. International World Wide Web Conference. Manhattan: ACM, 2004: 621-630

[3] Barbon F, Traverso P, Pistore M, et al. Run-time monitoring of instances and classes of web service compositions[C]//Leymann F, Zhang L J. Web Services. Chicago: IEEE, 2006: 63-71

[4] Martin D, Burstein M, Hobbs J, et al. OWL-S: semantic markup for web services[EB/OL]. Arlington: DAML Organization, 2003[2006-03]. http://www.daml.org/services/owl-s/index.html

[5] Dillon L D, Ramakrishna Y S. Generating oracles from your favorite temporal logic specifications[C]//Garlan D. Foundations of Software Engineering. San Francisco: ACM, 1996: 106-117

[6] Havelund K, Rosu G. Testing linear temporal logic formulae on Finite execution traces[R]. TR01-08, 2001

(责任编辑: 文丽芳)