面向方面分布式系统形式化规格说明语言*)

陈广明[1] 张立臣[2] 陈生庆[1]

(嘉应学院计算机系 梅州 514015)[1] (广东工业大学计算机学院 广州 510090)[2]

摘 要 分布式系统复杂性的不断增加以及对可配置性和可重用性要求的不断提高,需要面向方面软件工程方法的支持,而形式化方法能保证分布式系统的正确性。本文对分布式规格说明语言 Ocsid 进行了面向方面的扩展,讨论了面向方面的 Ocsid 的框架结构、语法要求、方面的联结和功能接口。定义了面向方面的 Ocsid 规格说明语言中叠加和组合的形式化描述,该形式化描述覆盖了各个精化阶段,使精化体系的各个独立视点被协调地组合,并能形式化地验证规格说明的时态属性和系统行为。本文的工作针对的是分布式系统的形式化规格说明,提出了面向方面 Ocsid 的形式基础和方面扩展,其基本思想同样适用于更一般的情况。

关键词 面向方面,分布式系统,形式化方法,Ocsid

Aspect-Oriented Distributed Formal Specification Language

CHEN Guang-Ming[1] ZHANG Li-Cheng[2] CHEN Sheng-Qing[1]

(Computer Department of Jiaying College, Guangdong, Meizhou 514015)[1]

(Computer College, Guangdong University of Technology, Guangzhou 510090)[2]

Abstract Increasing complexity of distributed system, and demands for enabling their configurability and reusability are strong motivations for aspect-oriented, and correctness of distributed systems requires that formal development methods are taken during software development cycle. This paper attempts to establish an aspect-oriented formal development method for distributed systems with the aspect oriented extension of Ocsid. The framework, syntax, weaving aspects, and interface of function are discussed in the paper. We present a formalization of how specifications are constructed using superposition and composition in the Ocsid specification language. The formalization covers stepwise refinement using superposition and composition of independent refinements. Independent views of a refinement hierarchy are reconciled. In the formalization. we can to construct formally verified temporal properties and action of distributed systems. The work that formalizaion and refactoring of Ocsid has been done in the context of aspect oriented formal specification of distributed systems, but we believe the ideas to be useful in a more general setting as well.

Keywords Aspect-oriented, Distributed system, Formal method, Ocsid

1 前言

面向方面的编程(Aspect Oriented Programming, AOP)吸取了面向对象开发的优点,开发者能关注于整个系统的方面特征的实现而无须了解其内在关系,最后通过联结器(weaver)实现系统各方面代码的结合。目前,从软件生命周期的宏观视角构建对 AOP 支持的 AOSD(Aspect-Oriented Software Department)是研究的热点。分布式系统固有的复杂性及软件对可配置性和可重用性要求的不断提高,需要面向方面软件工程方法的支持,将 AOSD 用于分布式系统的开发,能有效降低分布式系统开发的复杂性,提高可维护性和重用性。

诸如表达事件发生顺序或实时约束的时态属性(temporal properties)是系统正确性验证的基本标准[1],分布式系统中相对独立的各机制相互交织以及以效率为目标的各种优化措施使分布式算法难于理解和验证。形式化方法引入分布式系统的规格说明语言能对时态属性进行有效的推理,以确保系统行为的正确性。

支持面向方面的分布式系统形式化规格说明语言应满足如下的基本条件:(1)由于分布式行为涉及对象的互操作,语法单元应能在各抽象层面支持变量和操作的导入;(2)应支持横切关注点(crosscuting concerns)的分离和方面的(Aspects)的表示;(3)支持方面的联结(weaving)并在精化和联结过程中能保留其时态属性;(4)支持规格说明各层面正确性验证的一致性。

文[2]中提出的 AO-RT-Z(Aspect Oriented RT-Z)扩展了 RT-Z 使其支持 AOSD,文[3]中提出了支持面向方面的非形式化规格说明语言的基本构建原则。本文论述了一种典型的分布式系统规格说明语言 Ocsid 构建面向方面支持的形式化基础,通过对抽象语言要素的形式定义,在保留原有语法结构和语义的基础上,完成了方面扩展,其基本方法也适合于其它各种规格说明语言,具有一定的普遍性。

文中第 2 部分讨论了 Ocsid 语言的基本结构;第 3 部分提出支持面向方面开发的形式化基础;第 4 部分论述了面向方面 Ocsid 语言的结构支持,并通过一个简单实例说明其应

*)基金项目:本文受:1. 国家自然科学基金(No. 60474072、No. 60174050)、2. 广东省自然科学基金(No. 04009465、No. 010059)、3. 广东省高校自然科学研究项目(No. Z03024)基金资助。陈广明 副教授,硕士,主要研究方向:软件工程技术、形式化方法;张立臣 教授,博士,主要研究方向:分布式、实时系统;陈生庆 讲师,硕士,主要研究方向:软件工程技术、实时系统。

用;最后对全文进行了总结。

2 分布式规格说明语言 Ocsid

Ocsid 规格说明语言在 DisCo 语言[4]的基础上发展,该语言支持对分布式行为的验证。由于具有被 PVS 定理证明器的高阶逻辑给定的形式语义[5],该语言编写的规格说明能进行有效的形式推理。

类是构成 Ocsid 规格说明的基本结构,系统的状态由位于对象中的状态变量所描述[6]。Ocsid 语言通过叠加(superimpose)和行为交互(joint action)两种结构实现了规格说明的逐层精化,通过由线性时序逻辑定义的形式语义形成系统行为验证的基础。如果满足规格说明给定的初始条件,也即对象组合的卫士条件求值为真,行为处于可用状态(enable),如果有超过一个行为处于可用状态,系统非决定性地选择被执行的行为,行为的交互执行改变系统的状态变量,推进了系统的演化。

图 1 描述了在链表中进行节点删除的 Ocsid 规格说明。参与行为的类对象 toDelete 和 pred 被称之为角色(roles),通过这两个对象的交互行为 delete 完成了节点的删除,在分布式系统中其同步的具体实现,需要通过精化的规格说明予以表达。

```
specification list is
class node is
NEXT: ref node;
end;
action delete by toDelete,pred: node is
when pred. NEXT=ref(toDelete)do
pred. NEXT:=toDelete. NEXT;
toDelete. NEXT:=ref(toDelete);
end;
end;
```

图 1 删除节点的 Ocsid 规格说明

2.1 叠加

叠加是 Ocsid 语言中实现逐步精化的主要方式,它的语法非常简单,其作用是将新的规格说明内容增加到基本的规格说明中。Ocsid 叠加过程在类中导入新的状态变量,增加行为的卫士条件,并且对新导入的变量赋值。由于保留安全属性(preserves safety properties)的需要,先前导入的变量不能被重新赋值[7,8]。

叠加结构可以被重新定义为支持方面的语法结构,各个关注点描述的横切需求可以被封装在一个叠加层(layer)中,通过联结过程即完成系统各方面的结合,因此要求叠加具有面向方面支持的语义精确性,本文后续部分将对此进行讨论。

2.2 行为交互

行为交互是 Ocsid 语言规格说明支持分布式系统描述的基础。在基本的规格说明中包括类集合,连接行为集合和系统的初始条件。系统的状态取决于位于对象中的状态变量。行为交互定义了可参与该行为的角色集合,行为可以在任何对象组合的卫士条件为真时执行,当一个行为被执行,参与行为的对象状态将被改变,而系统的其余状态保持不变[9,10]。

3 方面扩展的形式化基础

3.1 Ocsid 语言的抽象表示

由于可读性的原因,Ocsid 语言书写的规格说明没有形式表示,为表达方便,给出 Ocsid 语言的形式化抽象描述,该描述具有等价性,不改变语言的语法特征和语义,由于不涉及当前的讨论,因此一些细节性的语法要素被忽略。通过对该抽象描述的形式定义,完成了对语言的方面扩展。specification 表示规格说明,class 和 action 分别表示类和交互行为,而 layer 表示叠加,它是精化和方面支持的基础。composition 完成了系统各部分的结合。

class(*names*, var*iables*)
action(*names*, *roles*, *guard*, *conjuncts*, *assignment*)
specification(*class hieratchy*, *action hieratchy*, *initial conditions*, *class extensions*, *action extensions*)
layer(*classes action*, *initial conditions*, *class derivations*, *class extensions*, *action derivations*, *action extensions*)
composition(*class mergings*, *action mergings*)

图 2 Ocsid 语言的抽象表示

3.2 派生和扩展的形式化表示

在面向对象的开发中,精化过程形成了程序实体的派生,可以通过关系来表示派生。

关系 R 的传递闭包表示为 R^*,组合符号⊕的定义为:

$$\varepsilon 1 \oplus \varepsilon 2 \triangleq (\varepsilon 1 \cup \varepsilon 2)^* \quad (1)$$

派生结构表示为三元组

$$H=(N,\varepsilon,R) \quad (2)$$

其中 N 是标识符的集合,ε,R 是标识符构成的关系。

关系 ε 是将标识符集合划分为等价类的等价关系。每一个等价类代表了规格说明的一个实体。关系 R 按照标识符记录了派生的历史。

实体的实际派生关系 D 被定义:

$$D \triangleq \varepsilon \oplus R \quad (3)$$

无派生关系 S 被定义为:

$$S(a,b) \triangleq \neg \exists c : D(c,a) \wedge D(c,b) \quad (4)$$

扩展结构是类和行为等实体的精化片段,类扩展结构由状态变量 var s 决定,其定义为:

$$structureC(n, H, X) \triangleq class(names(n, H), \bigcup_{x \subset applicable(n,H,X)} \text{var } s(x)) \quad (5)$$

行为扩展由角色 *roles*、卫士条件 *guard conjuncts*、和变量赋值 *assignments* 决定。其定义为:

$$structureA(n,H,X) \triangleq action(names(n,H), \bigcup_{x \in applicable(n,H,X)} roles(x), \bigcup_{x \in applicable(n,H,X)} conjuncts(x), \bigcup_{x \in applicable(n,H,X)} assignment\ s(x)) \quad (6)$$

实体的语法结构由派生结构集合和扩展结构集合共同决定,函数 *structureC* 返回类的结构,该结构表示与标识符 n 相关的派生结构集合 H 和扩展集合 X。函数 *structureA* 则返回交互行为的结构。

$applicable(n,H,X)$ 是在 X 中 n 派生的扩展集合:

$$applicable(n,H,X) \triangleq \{x \in X \mid D_H(n, name(x))\} \quad (7)$$

函数 *name*(e)返回扩展 e 代表的实体标识符,函数 *names*

(n,H)根据 ϵ_H 返回 N_H 中派生于 n 的标识符集合。其余函数 var s, *roles*, *conjuncts*, *assignments* 的基本含义同 *name* 类似。

3.3 Ocsid 语言方面扩展的形式化规范

3.3.1 规格说明的形式表示

要实现 Ocsid 语言对方面的支持,需要形式地表达语言的规格说明以利精确讨论。可以将 Ocsid 语言的规格说明定义为:

$$S=\text{specification}(H^c,H^a,I,X^c,X^a) \tag{8}$$

其中 H^c 是类名的派生结构,H^a 是行为名的派生结构,I 是初始条件的集合,X^c 和 X^a 分别是类和行为扩展的集合。与规格说明相关的实体结构由规格说明的派生结构和扩展过程决定。函数 *class* 和 *action* 返回同规格说明 S 中名字 n 相一致的实体结构。

$$class(n,S)\triangleq structureC(n,H_s^c,X_s^c) \tag{9}$$

$$action(n,S)\triangleq structureA(n,H_s^c,X_s^c) \tag{10}$$

3.3.2 叠加段的形式表示

叠加段用于表达规格说明的横切需求和并为精化提供支持,可以表示为元组:

$$L=\text{layer}(C,A,I,D^c,X^c,D^a,X^a) \tag{11}$$

C 是类的集合,A 是一个按照语法构造的行为的集合,I 是初始条件,D^c 是类派生二元组的集合,X^c 是类扩展集合,D^a 是行为派生二元组的集合,X^a 是行为扩展集合。其中派生二元组的形式为$(derived,base)$,它表示 *derived* 是由 *base* 派生而来。

3.3.3 联结过程的形式表示

在 Ocsid 语言中叠加结构建立了各抽象层面规格说明之间的无缝联结,而面向方面的联结过程同样是建立横切需求规格结构之间的无缝联结,因此在本质上叠加结构可以作为联结过程的基础。表示横切需求要求叠加段结构良好(well formed),实现联结过程要求叠加段具有封闭性(closing),因此叠加段应满足如下条件:

(a)叠加段的派生二元组仅仅派生 C 和 A 中的实体或者由叠加段导出的实体;

(b)叠加段的扩展仅仅扩展 C 和 A 中的实体或者由叠加段导出的实体;

(c)行为扩展的表达式仅仅涉及 A 和扩展导出的角色,并且仅涉及 C 或叠加段导出的实体的变量;

(d) 行为表达式仅指派叠加段导出的变量。

在结构 $H=(N,\epsilon,R)$ 上叠加一个派生二元组的集合定义为:

$$\text{sup erimpose}(D,H)\triangleq(N',\epsilon,R') \tag{12}$$

其中 $N'=N_H\cup\{d\mid(d,b)\in D\}$ $R'=R_H\cup D$

以定义(12)为基础在规格说明 S 上联结叠加段 L 的定义为:

$$\begin{aligned}&\text{sup erimpose}(S,L)\triangleq\\&\quad\text{specification}(\\&\qquad\text{sup erimpose}(D_L^c,H_S^c),\\&\qquad\text{sup erimpose}(D_L^a,H_S^a),\\&\qquad I_S\cup I_L,X_S^c,X_L^c,X_S^a\cup X_L^a)\end{aligned} \tag{13}$$

如果下列条件满足,联结后的规格说明将结构良好且具有封闭性,可以作为基本规格说明参与后续的联结过程。

(a)S 和 L 定义良好;

(b)派生和扩展不会引起实体标识符的冲突。

3.3.4 模块结合的形式表示

联结过程可以将不同关注点下的横切需求统一为一个整体。为了完成整个整体系统行为的验证和推理,还必须建立规格说明中各模块之间的接口,Ocsid 语言并未提供结构化的模块接口,因此需要形式地定义各模块组合的语义联系。

使用归并 M 结合各模块的集合 H 的定义为:

$$\text{compose}(H,M)\triangleq(N',\epsilon',R') \tag{14}$$

其中:$N'=(\bigcup_{h\in M}m)\cup(\bigcup_{h\in H}N_h)$

$\epsilon'=\text{buildMergeR}(M)\oplus(\bigoplus_{h\in H})\epsilon_h$

$R'=\bigcup_{h\in H}R_h$

归并 M 是一个标识符的集合,其元素为各个模块中表示相同实体的标识符函数 buildMergeR 返回一个等价关系,函数的定义如下:

$$\text{buildMergeR}(M)\triangleq(\bigcup_{m\in M}m\times m)^* \tag{15}$$

如果满足式(16)(17),模块的结合具有良好结构。公式(16)保证了在基本规格说明模块中表示分离实体的标识符不能表示结合后规格说明中相同的实体。公式(17)保证在基本规格说明模块中满足派生关系的标识符如果不表示相同的实体,则它们不能在最终的规格说明中表示相同的实体。

$$\forall h\in H:\neg\exists n_1,n_2\in N':S_h(n_1,n_2)\wedge D'(n_1,n_2) \tag{16}$$

$$\forall h\in H:\neg\exists n_1,n_2\in N':D_h(n_1,n_2)\wedge\neg\epsilon_h(n_1,n_2)\wedge\epsilon'(n_1,n_2) \tag{17}$$

使用 $C=\text{composition}(M_c,M_a)$ 结合规格说明集合 S 的过程可以定义为:

$$\text{compose}(S,C)\triangleq\text{specification}(H^{c'},H^{a'},I',X^{c'},X^{a'}) \tag{18}$$

其中 $H^{c'}=\text{compose}(\{H_s^c\mid s\in S\},M^c)$,$H^{a'}=\text{compose}(\{H_s^a\mid s\in S\},M^a)$

$$I'=\bigcup_{s\in S}I_s,X^{c'}=\bigcup_{s\in S}X_s^c,X^{a'}=\bigcup_{s\in S}X_s^a$$

如果下面的条件满足则结合后的规格说明具有封闭性和良好的结构:

(a)结合的模块具有良好的结构;

(b)扩展不发生命名冲突。

4 面向方面 Ocsid 语言的结构支持

以上通过形式化描述建立了 Ocsid 语言支持面向方面开发的基础,而形成符合形式规范的规格说明还需引入必要的语法结构。下面通过一个实例来概括性说明面向方面 Ocsid 语言的结构支持。

```
layer messages_1
requires
  class node is NEXT:ref node;end;
  action delete by toDelete,pred:node is
  when true do
    pred.NEXT:=…;
    toDelete.NEXT:=…;
  end;
provides
  new class request; new class reply;
  class extension node is …
    status(vaild,invalid);[status'valid].next:ref node;
  end
  class extension reply is …
    existsAs:(nothing,message)=…;
    [existsAs'message].to:ref node;[existsAs'message].next:ref
    node;
  end
  initially init is
        ∀  n:node;req:request;rep:reply::
           n.status=vaild^req.existsAs=nothing^rep.existsAs=
           nothing;
```

```
new action requestDelete;new action completeDelete;
action extension requestDelete by …n:node;r:request is
when …n. status=valid^r. existsAs=nothing do …
r. existsAs:=message'
r. [existsAs'message]. from:=ref(n);
r. [existsAs'message]. next:=n. [status'vaild]. next;
n. status:=invaild;
end;
action extension delete by …req:request;rep:reply is
when …pred. status=valid^rep. existsAs=nothing
  ^req. existsAs=message^req. [existsAs'message]. from=ref
  (toDelete)
do
  pred. [status'valid]. next:='pred. NEXT;
  req. existsAs:=nothing;
  req. existsAs:=message;
  rep. [existsAs'message]. to=ref(toDelete);
  rep. [existsAs'message]. next:='toDelete. NEXT;
end;
action extension completeDelete by …n:node;rep:reply is
when … rep. existsAs=message:=rep. [existsAs'message]. to=
ref(n)do
  n. status:=valid;
  n. [status'valid]. next:=rep. [existsAs'message]. next rep. ex-
  istsAs:=nothing;
 end;
end;
```

图 3　叠加段规格说明 messages_l

图 1 中的规格说明没有表达同步的实现方式，同时，在分布式系统中通过循环令牌实现交换请求、应答信息及行为协调是系统的横切需求。这些功能的由叠加段 messages_l 表示，系统的精化和方面的联结可以通过在规格说明 list 中联结叠加段 messages_l 实现。

图 3 中的叠加段在保留 Ocsid 语言中 layer 的基本结构的同时增加了对方面的支持，其中加粗标识符是语言的关键字。为支持公式(13)、(18)的要求，Layer 包含请求部分 requires 和提供部分 provides。requires 部分描述了为应用叠加而需要在基本规格说明中给出的类和行为，它是叠加段和基本规格说明联结的接口，provide 描述了联结附加在基本规格说明的新结构，extension 表示的结构将被叠加在基本规格说明的类和行为中。extensions 中的省略号"…"表示基本规格说明的类和行为。

在 Ocsid 语言中专门提供了表达状态变量可用性的结构，在定义了包含 vaild 和 invaild 两个值的枚举类型变量 status 后，定义变量 next 只有当变量 status 的值为 valid 时才能被访问的规格说明为[11,12]：

```
status:(valid,invalid);
[status'valid]. next:ref node;
```

在对 layer 验证时态属性时，隐含表示基本规格说明中有 layer 需求部分一致的行为[10]，当 layers 联结到规格说明时，联结语义暗含的条件需要显式地给出以支持形式推导。

$$
\begin{aligned}
&in\ variants \triangleq \\
&\quad \forall n \in node : n.status = valid \Rightarrow n.NEXT = n.[status'valid].next \wedge \\
&\quad \forall n \in node, r \in request : \\
&\quad r.existsAs = message \wedge r.[existsAs'message].form = ref(c) \\
&\quad \Rightarrow n.NEXT = r.[existAs'message].next \wedge \\
&\forall n \in node, r \in reply : \\
&\quad r.existsAs = message \wedge r.[existsAs'message].to = ref(c) \\
&\quad \Rightarrow n.NEXT = r.[existAs'messgae].next
\end{aligned} \tag{19}
$$

最终的 *layer messages_l* 联结到基本规格说明 *list* 的形式定义为：

$$
\begin{aligned}
&specification(list, messages\_l) \triangleq \\
&\quad superimpose(superimpose(list, in\ variants), messages\_l)
\end{aligned} \tag{20}
$$

最后简述应用令牌协调删除操作的实现过程。*layer messages_l* 提供了类 *request* 和 *reply* 以及行为 *requestDelete*，*completeDelete*。初始阶段有一个自由令牌在对象链表中传递，删除对象需要等待获取自由令牌，之后消费该令牌并发送一个保留令牌。然后完成删除操作。*layer messages_l* 保证了任意时刻只存在一个令牌并且只有一个对象可以完成删除操作。

总结　Ocsid 语言具有简洁的形式，能对面向对象开发中的各种结构和操作进行了精确表达，其清晰的语言特征有利于实现面向方面的扩展。Ocsid 语言根植于 PVS 定理证明器的支持，对其进行的面向方面的扩展能利用 PVS 定理证明器的推导过程完成系统行为分析。本文提出了 Ocsid 语言支持面向方面开发的形式基础，通过对抽象语言要素进行形式定义完成了规格说明语言的方面扩展，同时构造联结过程的语法结构，上述方法虽然针对 Ocsid 语言，然而对其它的分布式系统规格说明的方面扩展提供了形式化基础。

面向方面的软件开发方法需要在软件开发的各阶段实现横切需求的表达，对现有规格说明语言的方面扩展具有重要的价值，以形式化方法为基础的 Ocsid 语言方面扩展能利用语言固有的推理证明机制和辅助工具，保证分布式系统重要组件规格和设计的无二义性，方面联结的支持保证了系统模块分析的独立性、简洁性和正确性，为 AOP 提供了有力的支持。

参 考 文 献

1　Back R J R, Kurki-Suonio R. Distributed cooperation with action systems. ACM Transactionson Programming Languages and Systems, 1988, 10(4): 513～554

2　陈广明，张立臣，陈生庆. 面向方面的实时软件开发方法[J]. 计算机科学，2005，32(7)：189～192

3　陈生庆，张立臣，陈广明. 面向方面软件重构等价性的形式化证明方法[J]. 计算机科学，2006，33(7)：252～256

4　The AspectJ home page. At URLhttp://www. aspectj. org/, 2002

5　The DisCo project WWW page. Athttp://disco. cs. tut. fi on the World Wide Web, 201

6　Katz S, Gil J. Aspects and superimpositions. In: Position paper at ECOOP'99 AOP workshop, 1999

7　任洪敏，钱乐秋. 构件组装及其形式化推导研究. 软件学报，2003，14(6)：1699～1677

8　aki P K. A structural embedding of Ocsid in PVS. In: Richard J. Boulton and Paul B. Jackson, eds. Theorem Proving in Higher Order Logics, TPHOLS2001, number 2152 in Lecture Notes in Computer Science, Springer Verlag, 2001. 281～296

9　Cole L, Borba P. Deriving Refactorings for AspectJ. In: Proc. of the 4th International Conference on Aspect-Oriented Software Development (AOSD 2005), Chicago, USA, ACM Press, Mar. 2005

10　Andrews J H. Process-algebraic foundations ofaspect-oriented programming. In: REFLECTION'01: Proceedings of the Third International Conference onMetalevel Architectures and Separation of CrosscuttingConcerns, volume 2192, Springer-Verlag, Sept. 2001. 187～209

11　Gamma E, Helm R, Johnson R, Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994

12　Douence R, Motelet O, Sudholt M. A formal definition of crosscuts. In: REFLECTION '01: Proceedings of the Third International Conference on Metalevel Architectures and Separation of Crosscutting Concerns, volume 2192, Springer-Verlag, Sept. 2001. 170～186