◎博士论坛◎

服务本体驱动的构件协同模型

袁 援,王国胤
YUAN Yuan,WANG Guo-yin

重庆邮电大学 计算机科学与技术学院,重庆 400065
College of Science & Technology of Computer,Chongqing University of Posts and Telecommunications,Chongqing 400065,China
E-mail:yuanyuan@cqupt.edu.cn

YUAN Yuan,WANG Guo-yin.Coordination model of component driven by service ontology.Computer Engineering and Applications,2009,45(8):1-4.

Abstract: The critical issue for designing coordination among components lies in providing an observable interaction pattern for the components.Although this problem has been researched in many coordination models,they have not described their models at a common level.In this paper,services of the components are conceptualized with Description Logic,thus Service Ontology taken as observation benchmark of the components is built.After this ontology is applied in abstract Software Architecture,the connector in it is extended.Based on the above,the fundamental structure of coordination media is designed and a script-based coordination law is specified.A common coordination model of component is presented.

Key words: coordination model;component;service ontology;software architecture

摘 要:设计构件协同的关键在于向构件提供一种可观测的交互模式,众多协同模型虽然对此问题有所考虑,但并未将其抽象到通用级别。采用描述逻辑对构件服务予以概念化而构成服务本体,并将其作为构件可观测的基准,在软件体系结构的抽象级别上利用服务本体扩展了软件体系结构连接件。基于上述基本思路,设计了协同介质的基本结构并规范了基于脚本语言的协同规则,提出了一种通用的构件协同模型。

关键词:协同模型;构件;服务本体;软件体系结构
DOI:10.3778/j.issn.1002-8331.2009.08.001 **文章编号**:1002-8331(2009)08-0001-04 **文献标识码**:A **中图分类号**:TP303.1

1 前言

协同模型向软件系统中构件间的交互活动提供了一个高级别抽象。被普遍接受的观点是将协同模型看为协同实体(Coordination Entity)、协同介质(Coordination Media)和协同规则(Coordination Law)三个部分的组成[1],协同实体是协同活动的参与者,协同介质向协同实体提供交互活动空间,协同规则被封装于协同介质中以提供描述协同介质运作的规范。设计协同模型的一个重要问题是如何实现协同实体间透明的交互活动管理,即如何使提出协同请求的协同实体能在交互过程中确切定位其协同伙伴,此问题被称为协同的可观测性问题或可观测协同问题[2]。

集中式方法采用中央调度器实现协同请求实体对协同目标的寻找,例如 Bee-gent[3],但由于软件系统的逐步大型化,此方法已不太适合管理协同实体间复杂的交互活动;而分布式交互活动管理方法将中央调度器的功能分布到协同介质而以协同介质实现构件间的透明访问。COOL[4]是采用分布式方法管理交互活动的典型协同模型,它利用封装到协同介质中知识参考向协同实体提供了透明访问的解决方案,但它只能适用于封闭系统。Omicini 等将逻辑编程的思想引入协同模型,增强协同介质的推理能力,并力图基于元组中心的计算思想来推动开放环境中协同介质间的协同[5-7]。基于上述工作,MARS[8]、LuCe[9]、LIME[10]等模型提出了开放环境中 agent 之间的协同策略,并构建了驱动协同的知识。近年来,上述策略已逐步应用于语义 Web 环境中主体间的协同[11-12]。但这些工作大都是针对多主体协同的研究,且其中的协同知识只适用于具体应用,因而并不具有通用性。

对协同模型的研究是随着软件系统的分布化而出现的,软件系统的通用性体现于对分布式构件可重用性及系统动态重构的抽象,软件体系结构(Software Architecture,SA)则着力于此问题的探讨,因此,协同模型的通用性应该体现于软件体系结构级别的抽象;另一方面,体系结构描述语言作为 SA 的描述语言给出了构件的计算活动同其间交互活动的明确分离[13],这种分离不但使软件系统具有更好的可重用性、可配置性和可跟踪性,同时也满足了构件协同的基本要求,这是需要采用 SA

基金项目:国家自然科学基金(the National Natural Science Foundation of China under Grant No.60773113)。
作者简介:袁援(1972-),博士,讲师,CCF 会员,主要研究方向:分布式智能系统,语义 Web 服务;王国胤(1970-),博士,教授,主要研究方向:智能信息处理及应用。
收稿日期:2008-11-12 **修回日期**:2009-01-06

来刻画协同模型的另一原因。SA 的连接件虽然可很好地定义和管理构件间的相互连接，但却缺乏抽象构件服务功能的能力，而构件的服务能力在解决构件可观测问题上有其重要意义：构件服务是由构件直接或者间接向客户提供的功能，若以构件服务作为可观测协同的基准，协同请求构件找到的协同伙伴的计算能力将更接近客户需求。基于协同模型的基本概念，本文以 SA 作为描述协同模型的出发点，将构件服务抽象为服务本体并用其扩展了 SA 连接件，给出了一种服务本体驱动的通用构件协同模型。

2　基于服务抽象连接件的软件体系结构

2.1　协同模型的基本概念

在基本的协同模型中协同介质即为协同空间，本文将对协同介质予以扩展，而将协同空间作为协同介质的组成要素，下文的协同空间等同于基本协同模型中的协同介质。协同模型结构上的区分在于协同介质的差异：从协同空间的驱动方式来讲，协同模型分为数据驱动(Data-Driven)与控制驱动(Control-Driven)两类，二者的基本交互模式均为协同实体在协同空间上的数据发生和消耗，即协同请求实体是数据发生者，而数据消耗者则为协同伙伴，但它们在协同空间的基本工作方式上却迥然不同，前者是基于元组空间的数据交换，而后者则是基于通道的事件通知。考虑到事件通知机制是实现可观测协同模式的基础[14]，本文以控制驱动协同模型的通道作为本文协同模型的协同空间。协同模型内容上的差异则体现于协同规则的不同设计方案，多主体的协同强调协同协议的规范，其中的协同协议即为协同模型的协同规则。由于协同实体已预先存在于软件系统中，建模协同模型即是考虑协同介质和协同规则的设计。

2.2　体系结构描述

图 1 给出了基于服务抽象连接件的软件体系结构。同基本的 SA 相比较，它在连接件中引入了提供构件间可观测访问的实体——交互活动容器(Interaction Container,IC)，IC 为协同空间和规范构件抽象服务的本体提供运行环境，协同空间是实现构件交互访问的可重用体，而本体则为构件的可观测提供概念化知识描述，本文将此本体称为服务本体(Service Ontology,SO)，协同空间和 SO 在 IC 提供的运行环境中交互作用从而实现构件间的可观测访问。

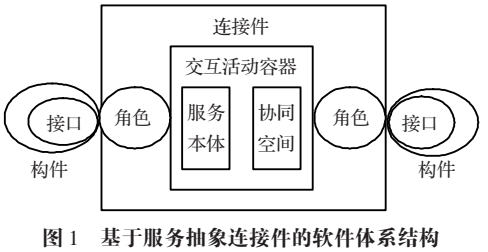

图 1　基于服务抽象连接件的软件体系结构

体系结构的其余组成部分描述为：

构件：体系结构中的计算实体，对应于协同模型中交互活动对象(协同实体)的集合；接口：构件方法的集合并定义了调用方法的协议，其基本功能是初始化并执行服务实例；连接件：提供构件交互活动的实体，由若干角色和一个 IC 构成，对应于协同模型中的协同介质；角色：角色提供了应用环境建模的抽象，它描述了构件在全局系统中执行任务的职责。

上述体系结构是一种基于事件隐式调用风格的 SA，构件间不是基于明确绑定而互访，而是采用事件通知模式进行触发式的、松耦合的、异步的交互。构件通过 IC 中的协同空间的事件触发互换信息，而其中的服务本体为体系结构中一定事件的发生和触发提供必要的知识；构件间可以彼此不知道对方的存在、不同时处于运行态而通信；构件运行时在体系结构中的添加、删除或修改不会影响其他构件的结构的行为。

2.3　服务本体的设计

本体是概念的明确规范，而概念则是关于客观世界的抽象、简化的视角。对 SO 而言客观世界就是构件服务，它为协同模型提供了如何组建构件服务的抽象化知识结构；另一方面，提供相同服务的异构构件可能对此服务有不同的表达形式，从此角度来讲，SO 也支持了异构构件间的服务概念意思的具体解释。因此，作为一个通用本体，SO 表达了构件服务的一般概念，为构件可观测提供了解决方案。SO 被设计为图 2 的层次化组织方式，动态服务描述了一些具有动态特性的服务，而静态服务则概念化了那些相对静态的服务，操作符服务指向应用环境的运行提供相关参数的构件服务，图中给出的是 SO 的高级分类，概念化服务可按客户的具体需求细分。

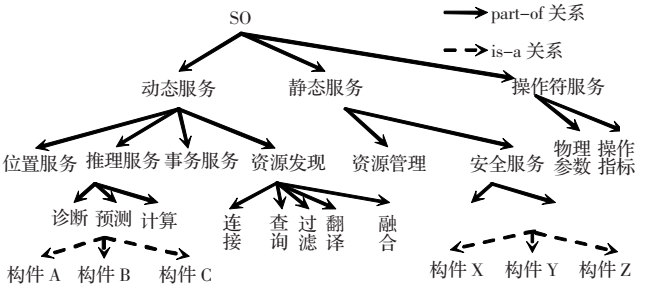

图 2　SO 的高级分类

SO 采用描述逻辑(Description Logic,DL)予以概念化描述。DL 是一种知识表示语言，它通过归类关系支持概念的自动层次化分类并向概念提供强有力的推理机制，由于具有此自动分类能力，DL 可增量式地规范概念。而 DL 将用户查询看作领域中的新概念，DL 系统(包括采用 DL 描述的领域及在其之上的查询)根据概念间的分类关系将此新概念自动定位到词汇的层次，用户可通过表达其信息需求而方便地查询领域知识。因此，DL 非常适合于 SO 的层次化概念组织方式的领域描述及在其之上的查询。

DL 根据概念、角色和个体来建模领域知识，具体应用中，它在两个级别上实现此创建：抽象级别规范领域对象的类(概念)及其间的关系(角色)，进行层次化组织；具体级别实例化抽象类的对象(个体)。下文在 DL 的抽象级别上创建 SO。不失一般，以下以 SO 中某个单一服务来阐述 SO 中服务概念的建立，采用 DL 对工作环境中诊断推理服务的描述如下：

(Define-concept DIAGNOSE_INFERENCE_SERVICE(AND SERVICE

(FILLS keyword "DiagnoseInferenceService")

(SAMEAS DIAGNOSE_INFERENCE_SERVICE(COMPOSE(SET symptom,Evidence)(SET ADD-Evidence,Delete-Evidence,Get-Conclusion)))

(THE model DIAGNOSE_INFERENCE_SERVICE-model)(ATLEAST 1 model)

(ALL options DIAGNOSE_INFERENCE_SERVICE-options)

(THE OwnedBy Component)(ATLEAST 1 OwnedBy)

(THE SerialNO INTEGER)(ATMOST 1 SerialNo)(ATLEAST 1 SerialNo)

(ALL history TRANSFER)))

由图 2 中的概念分类，推理服务概念 INFERENCE_SERVICE 是概念 DIAGNOSE_INFERENCE_SERVICE 的直接前驱，根据 DL 的分类关系,推理服务定义为:

(Define-concept INFERENCE_SERVICE(AND SERVICE(ALL has Parent DIAGNOSE_INFERENCE_SERVICE)))

对于诊断推理服务,可根据不同标准进行分类,例如,推理时间是评价推理构件工作能力的重要指标,下面的 DL 描述表示能够在 *MinIT*~*MaxIT* 时间内完成诊断推理的服务概念:

(Define-concept DIAGNOSE_INFERENCE_SERVICEII(AND DIAGNOSE_INFERENCE_SERVICE (ATLEAST *MinIT* Inference Time)(ATMOST *MaxIT* InferenceTime)))

整个 SO 采用上述 DL 描述方法建立概念及其层次化关系。

3　协同空间同服务本体间的交互

3.1　基本思想

在控制驱动的协同模型中,当协同请求者在协同空间上发生数据时将触发寻找协同伙伴事件。而对于本文模型,当此事件被触发时,协同空间必须首先访问 SO 以明确知道请求者的协同伙伴,这是体现此模型可观测性的根本。但 SO 是一静态的概念化实体,不能接收协同空间的事件通知,协同空间只能采用查询方式访问概念化实体 SO,因此,如果要实现协同空间同 SO 间的交互必须设计一种事件同本体查询间的相互转化机制。为此,本文在 SO 和协同空间之间引入一个实现此转化功能的引擎,称为协同管理器,其基本部件为 DL 解析器和 DL 封装器,DL 解析器解析来自于协同空间的事件以将其转化为 DL 查询,而 DL 封装器的功用为将从 SO 查询得到的结果封装为在协同空间之上的事件。

3.2　协同规则

基于上述基本想法，需设计的协同规则应该可转化为 DL 查询格式,本文以脚本语言来设计协同规则,这主要是基于下述考虑:脚本语言的嵌入式表达能够方便地将嵌入其中的事件转化为查询;从支持构件协同的角度来讲,脚本语言采用陈述式规范紧密地将软件构件穿插(plug)到一起,由此,采用脚本语言设计协同规则可保证构件具有非常可靠的协作关系。本文的脚本协同规范语言称为事件出现规则语言(Events Emerging Rule Language,EERL)，它是一个将构件的行为规范从需要协同此行为的通信中分离出来的协同脚本,其 EBNF 符号描述为:

rule:"on" pexpr[guard] "{" actions "}"

pexpr:"event"comp"("[ident(","ident)*]"("pexpr")"|pexpr op pexpr

guard:"not"*("in" | "out")"context" string

actions:[action(";"action)*]

comp:ident ":" ident

op:"where" | "and" | "or" | "eq" | "neq"

action:("call" | "emit")ident "("[expr(","expr)*] ")"

EERL 的基本表述方式为 On event *C*:*E*,它表示协同空间等待构件 *C* 的事件集合 *E* 出现的规则,若此事件出现,协同空间则传递一定的事件通知,否则,表示构件保持当前状态。因此,EERL 相当于以脚本的方式来表示在协同空间上事件的触发规则,若在协同空间中封装一定上述事件出现规则,则构成了本文协同模型的协同规则。在 EERL 的事件出现规则中,事件 *E* 的参数值采用 WHERE 子句进行检查,而其相关内容则通过 in context 或 out context 子句予以检查。

下面的 EERL 描述给出了协同空间等待最大推理时间为 *MaxIT* 的诊断推理构件 DIC1 的事件 Collative(DIAGNOSE_INFERENCE_SERVICE,WantMaxIT,Task)的出现规则。具体阐述为:假如 DIC1 推理时间已超过 1 s 且其当前任务是安全的,它发出一个查询服务本体以得到最大推理时间为 *WantMaxIT* 的事件并要求查询得到的构件接续其推理任务，而后调用函数 EndInference(self)终止其当前推理。

ON EVENT DIC1:Collative(DIAGNOSE_INFERENCE_SERVICE, *WantMaxIT*,Task)

WHERE(*InferenceTime*>*MaxIT*)

IN CONTEXT "secure"

{GoalComponent=EMIT QueryComponent(DIAGNOSE_INFERENCE_SERVICE,"*InferenceTime* at most",*WantMaxIT*);

EMIT QueryComponentIdle(GoalComponent);

IF QueryComponentIdle(GoalComponent)THEN

{EMIT TransferTask(GoalComponent,Task);

CALL EndInference(self);} }

3.3　EERL 同 DL 间的转换

EERL 以构件的方式表达查询事件通知，而 SO 的查询描述则是 DL 表达式,两种不同表达间的转换由协同管理器来完成,DL 解析器解析 EERL 脚本以执行本体查询,而将查询结果转化为构件可理解的 EERL 脚本的工作由 DL 封装器来实现。限于篇幅,以下只给出 DL 解析器的解析过程,而后一过程则是其逆运算。

在采用 DL 描述的本体之上进行查询的 DL 通用表达式为:

<list-of-roles> for getall <list-of-constraints>

其中,list-of-roles 是用户请求角色的列表,list-of-constraints 是采用 DL 表示的约束列表，表示了查询回答必须满足的条件。假如 list-of-roles 为空,查询返回特殊构件 self,表示寻找协同伙伴失败。

例如,对于查询最大推理时间为 *MaxIT* 的诊断推理服务可建立如下 DL 查询表达式:

[ReferenceComponentSerialNo] for getall(AND DIAGNOSE_REFERENCE_SERVICE(ATMOST *MaxIT* InferenceTime))

它是上节 EERL 代码中 EMIT QueryComponent(DIAGNOSE_INFERENCE_SERVICE,"*InferenceTime* at most",*MaxIT*)事件的 DL 查询表达形式。

将 EERL 脚本表示的查询事件解析为 DL 查询的过程事实上是设计将 EERL 查询事件中的词汇翻译为 DL 查询词汇的方案,其目标是获得以服务本体词汇表达的查询的同时保持 EERL 脚本查询事件中词汇语义的不变性,此过程采用在词汇间的同义词关系和值之间的等价关系而获得,而同义词由一个同义词汇库提供。翻译算法(TRANSLATER)如下:

TRANSLATER(EERL-script-query,SO)

FOR each constraint in EERL-script-query DO

CASE constraint is:

term:IF ∃synonym from a term in EERL-script-query to a term in SO THEN

{Substitute(term,synonym-of-term)}

ELSE IF the term in EERL-script-query is a new defined term THEN

{Substitute(term,definition-of-term);

Translate(definition-of-term,SO)}

value:IF ∃a transfer function between relation and new relation THEN

Substitute(value,equivalent-value)

4 协同介质的设计

根据上文的描述,本文协同模型的协同介质有如图3的整体结构。通道为协同空间的实现体,其中封装了EERL协同规则;协同管理器由DL解析器、DL封装器、同义词库以及本体查询接口组成,本体查询接口是协同管理器同服务本体交互的口岸;服务本体提供系统构件相互观测的概念化知识。

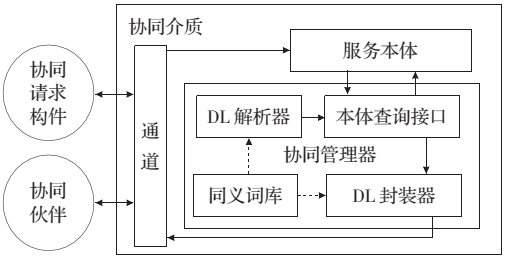

图3 协同介质的设计结构

实现构件间可观测协同的基本过程为:协同请求构件向通道发放协同请求通知,以EERL表示的本体查询事件被触发;此事件通知被发送到DL解析器并被其处理为DL查询,DL解析器将此DL查询转发至本体查询接口以实施查询;在查询到满足协同请求构件所需服务的构件后,服务本体向协同管理器返回查询结果;查询结果通过本体查询接口转发DL封装器,DL封装器将这些结果封装为事件通知并传送到协同空间;接到此事件通知后,通道向查询到的所有构件发送协同请求,若某构件空闲它即为协同请求构件的协同伙伴,它即消耗通道中的数据并计算,若所有查询到的构件都处于工作状态,通道等待这些构件之一空闲状态的出现,若此构件完成当前工作则消耗通道中的数据并计算,计算结果经通道返回协同请求构件。

5 结论及展望

作为协同模型的范畴,可观测模式出现于数据库、多主体系统、多媒体系统等不同的研究和应用领域,因此,它反应了软件系统的普遍需求。本文通过采用服务本体扩展基本软件体系结构的连接件提出了一种概念化的构件可观测协同模型,同COOL、MARS、LuCe、LIME等基于知识解决协同可观测性的协同模型相比,此模型的最大特点为其适应不同应用环境的通用性,这体现于:

(1)软件体系结构是抽象建模分布式软件系统的重要方法,将描述分布式构件间协同的模型抽象到软件体系结构级别为构件协同提供一个高级别抽象;

(2)将构件服务抽象为服务本体为构件间的可观测协同提供了一个概念化的解决方案,服务本体不但概括了全局系统中的构件服务而且为其提供了一种语义级别的、统一的表达;

(3)基于脚本的协同规则的设计为协同模型提供了一种泛化的协同规范,文中协同模型也支持了此协同规则概念同服务本体概念间的转换。

Mediator是协同介质的实例化实体,众多分布式软件系统以其作为协同构件的协同介质[15],针对本文的协同模型也可将其协同介质实例化为Mediator,若协同实体为multi-agent,底层通信设施为实际应用的网络环境,则多主体系统的协同是文中模型的一个应用实例。今后将主要致力于以下工作:描述底层通信设施的通信机制;将此协同模型应用于多主体系统的实际开发。

参考文献:

[1] Ciancarini P.Coordination models and languages as software integrators[J].ACM Computing Surveys,1996,28(3):300-315.

[2] Antonio Brogi,Jean-Marie Jacquet.On the expressiveness of coordination models[J].Coordination Languages and Models,1999,15(2):134-149.

[3] Kawamura T.Bee-gent:Bonding and encapsulation enhancement agent framework for development of distributed systems[J].Journal of IEICEJ,1999,8(9):735-751.

[4] Barbuceanu M,Fox M.COOL:A language for describing coordination in multi agent systems[C]//Proceedings of the First International Conference on Multi-Agent Systems.[S.l.]:AAA Press,June 1995:17-25.

[5] Omicini A,Zambonelli F.TuCSoN:From tuple spaces to tuple centres[J].Internet Research:Electronic Networking Applications and Policy,1998,8(5):400-413.

[6] Omicini A,Zambonelli F,Coordination for Internet application development[J].Autonomous Agents and Multi-Agent Systems,1999,2(3):251-269.

[7] Ciancarini P,Tolksdorf R,Vitali F,et al.Coordinating multiagent applications on the WWW:A reference architecture[J].IEEE Trans on Software Eng,1998,24(5):458-475.

[8] Cabri G,Leonardi L.MARS:A programmable coordination architecture for mobile agents[J].IEEE Internet Computing,2000,4(4):26-35.

[9] Denti E,Omicini A,Toschi V.The LuCe coordination technology for MAS design and development on Internet[C]//Proceedings of the 4th International Conference on Coordination Languages and Models.London,UK:Springer-Verlag,2000:305-314.

[10] Murphy A L,Picco G P,Roman Gruia-Catalin.LIME:A coordination model and middleware supporting mobility of hosts and agents[J].ACM Transactions on Software Engineering and Methodology(TOSEM),2006,18(7):613-637.

[11] Tafat A,Courant M,Hirsbrunner B.A generic coordination model for pervasive computing based on semantic Web languages[C]//International Conference on Applications of Natural Language to Information System,Salford,2006:265-275.

[12] Tolksdorf R,Bontas E P,Nixon L J B.A coordination model for the semantic web[C]//Proceedings of the 2006 ACM Symposium on Applied Computing,Coordination Models,Languages and Applications,2006:419-433.

[13] Papadopoulos G,Arbab A.Configuration and dynamic reconfiguration of components using the coordination paradigm[J].Future Generation Computer Systems,2001,17(8):1023-1038.

[14] Murillo J M.Tuple-based coordination models in event-based scenarios[J].Coordination Languages and Models,2002,15(4):53-68.

[15] Wiederhold G.Mediators in the architecture of future information System[J].IEEE Computer,1998,25(12):40-52.