基于移动 Agent 的反射式异构服务协同机制初探*)

胡海洋　葛季栋　马晓星　吕　建
(南京大学计算机软件新技术国家重点实验室　南京 210093)
(南京大学计算机软件研究所　南京 210093)

摘　要　使用不同类型的中间件实现了开放动态环境中的软件服务。客户方的应用需要采用灵活的方法来发现并绑定这些服务。本文提出一种基于移动 agent 的反射式异构协同机制，它通过动态加载功能构件可重配置 agent 当前对外协同行为，通过动态创建子协同 agent 可实施多种服务发现机制与服务绑定机制的并发执行，可有效减轻客户端应用的负担，提高对异构服务的协同效率。

关键词　移动 agent，软件协同，反射式

A Reflective Service Coordination Mechanism Based on Mobile Agents

HU Hai-Yang　GE Ji-Dong　MA Xiao-Xing　LU Jian
(State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093)
(Institute of Computer Software, Nanjing University, Nanjing 210093)

Abstract　Software services in open environment are implemented using different middleware types and advertised using different discovery protocols. Client application needs to adopt a flexible approach to discover and bind these services. This paper presents a reflective coordination mechanism based on mobile agent. It dynamically adapts both its binding and discovery protocol to allow interoperation with heterogeneous services by uploading different functional components. By dynamically creating multiple agents at run time, it can interoperate with multiple services in parallel, and as a result the client burden is released and the efficiency of the coordination process is enhanced.

Keywords　Mobile agent, Software coordination, Reflection

1　前言

Internet 技术的发展提供开放、动态和多变的环境，运行于其上的软件具有自主性、协同性、多态性等特征[1]。对开放动态环境中的软件服务进行协同调用已为目前研究者所关注。在协同调用过程中，需要有效的服务发现机制来发现环境中存在的软件服务，进而对这些服务进行绑定，并实施相关的方法或过程调用。因此服务发现与绑定机制在软件服务协同调用研究中占有重要的地位。

对服务发现与绑定机制的研究由来以久，Jini[2]、UPnP[3]、SLP[4]等均实现了不同服务发现系统，面向对象中间件系统[5~7]也提供了发现与绑定软件服务的功能。上述系统在实际应用时，一般需要客户应用方与服务方均基于同一系统或协议开发而成，而不能有效支持跨系统的协同调用。如采用 UPnP 发现方式的应用程序不能发现基于 SLP 协议发布的服务；中间件系统 CORBA 尽管支持对异构服务的协同调用，但也需客户应用程序与软件服务均采用 CORBA 的封装与实现方式，而不能支持跨中间件平台的服务调用。因此在开放环境中对于采用单一服务发现与绑定机制的客户应用而言，其可错失发现与调用其他类型软件服务的契机，而该方面的需求在移动应用环境中显得尤为重要。

目前反射式中间件系统在此方面做了较多努力，它们通过动态重配置自身的结构行为来支持对多类型异构服务的协同调用。如 UIC[8]可同时对多种中间件类型软件服务的绑定调用，ReMMOC[9]可同时支持多种服务发现协议，及对多种类型异构服务的绑定调用。这些系统一般均建立在构件[10]技术基础上，内部存储多个功能构件，为各种服务发现与绑定协议的具体实现，通过加载不同功能构件以实施对多种类型服务的发现与绑定调用。对上述反射式中间件系统分析后可发现，功能构件的存储、动态加载均由客户方完成，因此客户端程序需要频繁地动态重配置自身结构行为，这对于某些硬件能力有限的客户端(如移动设备)将较难承受；同时系统在发现服务过程中，客户应用轮流启动各种服务发现功能构件，不停轮询当前环境，直至找到服务为止，该过程也将消耗客户端较多资源；而且对于多种异构服务的并发调用，这些系统也不能有效支持。上述方面使得它们不完全适合开放环境中软件服务动态协同调用的需求。移动 agent 技术为解决上述问题提供了良好的技术基础[11,12]。本文给出一种基于移动 Agent 的反射式异构服务协同机制，来实现客户端应用的轻量化(light weight)，避免客户应用频繁的重构自身配置，该机制可同时支持多种服务发现机制，并可实施对多种异构服务的并发调用，从而满足开放环境中异构服务动态协同调用的需求。

本文在介绍 agent 反射式服务协同机制基本结构的基础

*)本课题得到国家自然科学基金(60233010，60273034)、国家"九七三"重点基础研究发展规划项目(2002CB312002)、国家"八六三"高技术研究发展计划项目(2002AA116010)资助。胡海洋　博士生，主要研究领域为构件、中间件技术。葛季栋　博士生，主要研究领域为软件过程。马晓星　博士，副教授，主要研究领域为软件体系结构、Internet 软件技术。吕　建　教授，博士生导师，主要研究领域为软件自动化、并行程序形式化方法、面向对象语言和环境。

上，阐述了相关的 agent 重配置过程和运行机制，最后对相关工作做了小结与比较，并进行了性能测试。

2 基本框架

反射概念[13]起源于人工智能领域，它是指如果一个计算进程含有这样一个成分描述进程，该进程能够描述计算进程其自身的操作和数据结构，则该计算进程也能推理其自身。在反射系统的实现中，一般采用“关注分离[14]”（separation of concerns）的原则，即系统分为基层和元层，基层一般用于对具体问题领域的抽象，元层则是对基层及系统内部的表示，并且与其因果相连，通过对元层的调整可实现系统结构行为的动态重配置。反射系统具有开放的结构，可适应环境动态变化的需求。

基于移动 agent 的反射式服务协同框架（图 1）中，客户方通过 agent 实施对环境中软件服务的协同调用，移动 agent 代替客户端完成对环境中软件服务的发现与绑定，以减轻客户端应用的负担；功能构件是一组封装良好的实体，为不同服务发现与绑定方式的具体实现，它构成了元层的基本元素，移动 agent 可加载不同功能构件以重配置自身对外的协同行为；agent 也可从环境节点中下载新型功能构件，动态扩充自身对外协同能力，从而完成对新类型服务的协同调用。

图 1 基本框架

3 基于 agent 的协同机制

在反射式服务协同机制中，对异构服务的发现和绑定均由移动 agent 完成，具体而言，系统中有应用 agent 和协同 agent 两类组成，前者表示一次具体的服务协同应用，后者则用于最终发现和绑定服务实体。

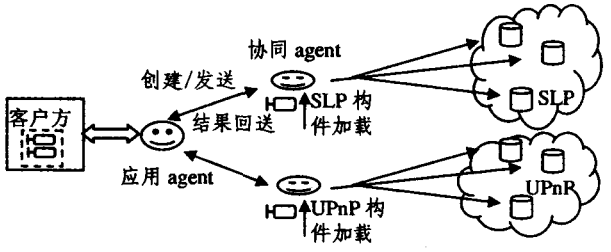

图 2 应用 agent 与协同 agent 运行示意

客户应用进行服务协同时（图 2），当向环境派发出一个应用 agent，其携带着客户应用具有的功能构件，完成对环境中软件服务的搜索与绑定调用，并将最终结果返回给客户端。整个过程中，客户端仅发送与接收应用 agent，其负载大为减轻。对环境中服务的发现，应用 agent 可以轮流启动多个服务发现功能构件，不断轮询环境直至其中发现所需服务为止，但这样同样将消耗应用 agent 较多资源，且需不停动态加载功能构件并重配置内部结构行为，从而影响其运行效率。本文采用了基于 agent 动态分解的做法，即应用 agent 并不直接进行服务的发现与绑定，而是动态创建多个协同 agent，每个协同 agent 分配不同类型服务发现构件与服务绑定构件，这些协同 agent 并发运行于环境中的不同节点上用于发现与绑定环境中不同类型的软件服务，协同 agent 将协同结果返回给应用 agent，应用 agent 可向协同 agent 传送新的功能构件以重配置其对外协同行为。

应用 agent 内部结构中（图 3）包括功能构件表、协同 agent 表、服务管理、客户应用交互接口、协同 agent 交互接口等几部分。功能构件表存储该应用 agent 具有的功能构件，包含服务发现与服务绑定两类；应用 agent 动态创建的协同 agent 则记录在协同 agent 列表中，相关记录信息还包括协同 agent 当前状态、所分配的功能构件类型、所在物理位置等；服务管理用于动态创建、派发新协同 agent、协同 agent 的注销、协同结果的挑选等；应用 agent 对外接口有两种：客户应用接口和协同 agent 交互接口，前者用于与客户应用程序进行通信，如协同结果传收、新功能构件的传收、客户端新服务协同请求的传收等，后者与自身创建的协同 agent 通信，包括新功能构件的传收、协同结果的传收、状态信息的传收、新协同需求的传收等。

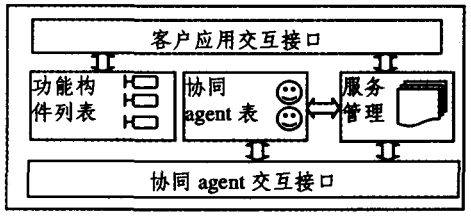

图 3 应用 agent 内部结构

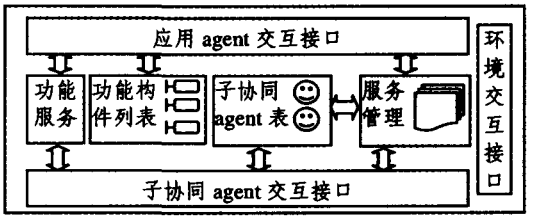

图 4 协同 agent 内部结构

协同 agent 内部（图 4）包括功能服务、功能构件列表、子协同 agent 表与服务管理。功能服务为协同 agent 当前实施的服务发现功能与服务绑定调用功能，协同 agent 含有一特定类型的服务发现功能构件，用于发现环境中基于该服务发现协议而发布的软件服务，其还含有若干类型的绑定功能构件，用于绑定调用所发现异构服务；开放环境中可存在大量提供相同功能的异构软件服务，协同 agent 发现这些服务实体后，可以通过动态加载不同的服务绑定功能构件以分别去调用这些服务，此时其需多次重配置自身对外的服务绑定结构行为，协同 agent 也可动态创建多个子协同 agent，并为每个子协同 agent 加载不同类型的绑定功能构件，使其可绑定调用特定类型的软件服务实体，以实现对多种类型异构服务的并发调用；协同 agent 可通过交互接口与应用 agent 通信，相关内容包括传送服务协同调用结果、自身状态信息等；其也通过环境交互接口与环境进行交互，从环境中下载新型功能构件，以动态扩充自身协同能力。

4 协同 agent 的动态重配置

协同 agent 在实施服务发现与绑定调用过程中，可通过加载新功能构件、动态重配置自身结构行为来适应开放环境的动态变化性。其动态重配置过程(图 5)包括传送、归纳、合成、运行步骤：1)传送。应用 agent 可向协同 agent 传送新功能构件，对其协同行为进行动态调整，协同 agent 也可向其创建的子协同 agent 传送新功能构件。2)归纳分析。协同 agent 接受外界传送功能构件后，将对其进行归纳分析，分析方面包括：(a)agent 能否识别该功能构件。一般说来，新构件与被替换功能构件应具有相同服务接口，agent 可直接识别启动，若新功能构件实现了某些新服务接口，则相关接口文件(如 WSDL)也应同时被传送；(b)被替换功能构件当前的状态。若被替换功能构件目前正与外界协同交互，此时无法实施替换，需待其相关交互行为终止后再行替换。3)合成。功能构件与协同 agent 内部其他成分之间存在着一定的参数传递及接口调用关系，新加载功能构件通过合成步骤将重新建立并维持这些关系结构。4)运行。上述步骤完成后，协同 agent 将运行该功能构件测试其正确性，若运行过程有误，协同 agent 将重新换回被替换功能构件，并向相关传送者通报出错信息，等待其再次传送新功能构件。

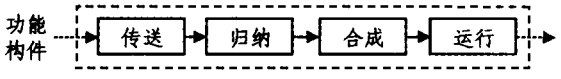

图 5 协同 Agent 动态重配置

构件通过接口进行连接，为了调和功能构件之间互操作，及减轻 agent 重配置过程的复杂度，功能构件之间并不直接相连接口，而是通过特制适配器(Adapter)连接起(图 6)。适配器是一个具有多个接口的实体，用于连接多个构件，其从某个接口获取数据并进行内部处理后根据需要向其他接口发送，适配器内部处理工作包括类型匹配、转换、数据过滤、数据挑选等工作，可屏蔽构件接口之间的失谐。协同 agent 进行构件重配置时，通过适配器探知被替换构件当前状态，构件替换后，新功能构件将建立起与适配器的连接，而其他功能构件与该适配器的连接则无需改变，这样避免了构件的直接连接，有助于降低构件之间的耦合度，同时对于某些构件接口不匹配的情形，也可通过适配器内部数据类型的处理与转换等工作加以解决。客户应用也可通过加载新的适配器，调整功能构件之间的连接行为，如进行新的数据过滤算法、转换工作等。

图 6 功能构件的连接

对于开放环境中发布的一些新型异构服务，其可将自身特定服务发现与绑定方式封装成功能构件形式放置在环境节点中以供外界应用访问使用；外部应用协同调用这些服务时，需先下载并加载相关功能构件。此类功能构件发布时，相关接口文件也需被同时分布，使得外部应用可对其有效识别并运行。协同 agent 可从环境中动态下载这些功能构件，并可向其他协同 agent、应用 agent、客户端设备传送，从而动态扩充整个客户应用发现与绑定异构服务的能力。对于一些硬件能力有限的移动计算设备，其自身无法存储过多功能构件，通过 agent 从环境节点中动态加载功能构件，可有效减轻其负担。

5 协同 agent 的运行

开放环境中可存在大量提供相同功能的异构软件服务，协同 agent 发现多个此类软件服务时，可采用不同运行方式进行绑定调用。

顺序绑定调用方式中，协同 agent 将依次调用所有服务，最后从中选择结果最适合者，在此过程中，agent 将多次动态加载各种服务绑定功能构件，则协同 agent 在顺序方式中调用异构服务所需时间为：

$$\sum_{i=0}^{n-1}(T_M(L_i,L_{i+1})+T_{RC}(\mathrm{Comp}_{i+1}))+T_S(S_{i+1})+T_{SEL}(SR) \quad (1)$$

其中 $T_M(L_{i-1}, L_i)$为协同 agent 迁移至服务 S_i所在物理节点 L_i所需时间；$TS(S_i)$为 agent 调用软件服务 S_i 所需的时间；$T_{RC}(\mathrm{Comp}_i)$为协同 agent 加载与 S_i 相适应功能构件 Comp_i 并实施重配置并所需时间；$T_{SEL}(SR)$为 agent 从结果集 SR 中选择最理想结果所需的时间。

协同 agent 也可支持对多种异构服务的并发调用，在此方式中(如图 2)，协同 agent 在某节点 L_0 上动态生成若干子协同 agent 并为这些 agent 加载不同类型的功能构件，然后实施并发调用多个服务，待所有协同 agent 返回后，将从结果集中选择最适合者，此并发方式所需时间为：

$$T_C(n)+\mathrm{Max}_i(TM(L_0,L_i)+TS(S_i)+TM(L_i,L_0))+T_{SEL}(SR) \quad (2)$$

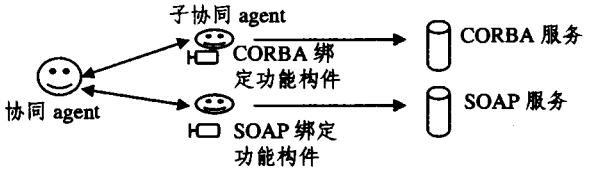

图 7 协同 agent 并发绑定调用方式

其中 $T_C(n)$为创建 n 个子协同 agent 所需时间，$TM(L_i, L_0)$为子协同 agent 返回 L_0 所需时间。

环境中若存在多个可下载功能构件的物理节点时，协同 agnet 也可采取顺序和并发两种运行方式去获取功能构件以扩充自身对外协同能力，两种方式各自所需开销类似式(1)与式(2)。

6 性能分析

本节进行了性能测试和模拟实验。在实验中用到了两种环境配置：1 台 IBM NetVista 1.7GHZ/Windows XP Professional 用于客户段创建、派发和接受 Agent；数台 Dell PowerEdage 1400SC 2.0GHZ/Read Linux 8.0 服务器用于放置环境中的软件服务。服务器之间通过 100M 以太网连接。

一些基本开销的测试如表 1 所示。其中 agent 从一个物理节点迁移至另一物理节点的开销，包括 agent 对象实例传送及 Class 文件传送两部分。agent 动态创建的开销主要包含子协同 agent 对象的动态创建、功能构件加载和进行相关设置的开销。协同 agent 动态加载功能构件所需开销包括，

(下转第 171 页)

结论 模糊概念的提出使得人们表达事物更精确，更广泛也更有针对性。从这一点上讲，Vague 集强调了肯定与否定的重要性。对于 Chen 所提出的关于 Vague 集的相似性度量 Hong 的否定方法是错误的。需要指出的是，不同度量方法刻画事物不同的方面。Hong 和 Chen 的这两种度量方法都有实际应用意义。

参 考 文 献

1 Zadeh L A. Fuzzy sets. Inform. And Control, 1965, 8: 338～356
2 Gau W L, Buehrer D J. Vague sets. IEEE Trans. on Systems, Man, and Cybernetics, 1993, 23(2): 610～614
3 Chen S M. Measures between vague sets. Fuzzy Sets Syst., 1995, 74(2): 217～223
4 Chen S M. Similarity measure between Vague sets and elements. IEEE Trans. on Systems, Man, and Cybernetics, 1997, 27(1): 153～158
5 Hong D H, Kim C. A note on similarity measures between vague sets and between elements. Information Science, 1999, 115: 83～96
6 闫德勤，迟忠先，李艳红. 关于 Vague 集的相似度量. 模式识别与人工智能，2004，17(1)：22～26

(上接第 148 页)

将字节流形式的功能构件转化为类文件、生成功能构件的对象实例、确定方法入口和进行相关设置与配置。

表 1 一些基本的开销测试

相关操作	所需开销(ms)
Agent 从一个节点迁移至另一个节点	约 50ms
动态创建子协同 Agent	约 2ms
动态加载功能构件	约 3ms

图 8 测试两种获取方式的开销。当随着访问节点数的增多，顺序获取方式的开销将明显超过并发获取，这是因为顺序方式中 agent 在节点之间迁移所需开销较大，而并发方式中动态创建多个子协同 agent 所需开销有限，因此当协同 agent 需在较短时间从多个节点下载功能构件时，可采用并发运行方式。图 9 比较了顺序调用方式与并发调用方式的开销。可以看出当协同调用任务数<3 时，并发方式并不占优势，这是由于节点需创建多个子协同 agent 并发送出去，当 agent 所需调用的软件服务个数较少时，发送多个 Agent 的开销就占了相当的比例，但随着协同调用软件服务个数的增加，并发调用的优势就比较明显，这是因为顺序方式中 agent 需在节点间多次迁移，此外其还需多次动态加载功能构件并进行配置。

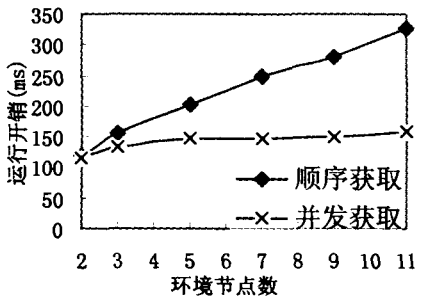

图 8 不同获取方式比较

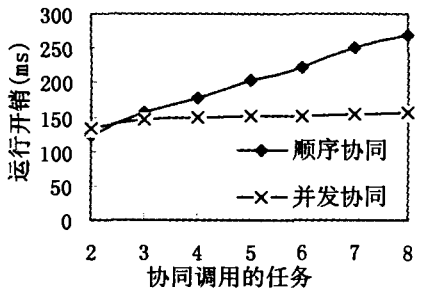

图 9 不同运行方式比较

7 相关工作的比较及结论

近年来，研究者开发了具有自适应能力的反射中间件系统，用以在开放环境中实施灵活的异构服务协同机制。本文现对这几种反射中间件比较如表 2 所示。

表 2 几种反射式异构服务协同机制的比较

	反射粒度	重配置实体	协同功能动态扩充	异构服务绑定调用机制	服务发现机制
ReMMoC	构件	客户端应用	不支持	不能同时支持多类型	多种机制轮询
UIC	构件	客户端应用	不支持	可同时支持多种类型	不支持
基于 agent 的协同机制	构件	协同 agent	支持	多 agent 并发	多 agent 并发

从上表可以看出，本文采用基于 agent 的反射式异构服务协同机制通过动态创建协同 agent 及加载功能构件避免客户端频繁动态重配置自身结构行为，且可并发实施多种类型服务发现与绑定机制，有效地提供了协同效率，并减轻客户端负担，实现了客户端应用的轻量化，同时该机制可支持系统协同功能的动态扩充，因此其较为适应开放环境动态变化的需求。

本文提出了一种基于移动 Agent 的反射式异构服务协同机制，其通过动态加载功能构件可重配置 agent 当前对外协同行为，通过 agent 动态创建子协同 agent 可实施多种服务发现机制与服务绑定机制的并发执行，可有效减轻客户端应用的负担，提高对异构服务的协同效率。

今后的工作包括对移动 agent 实施反射过程中的安全性维护、构件组合形式化验证研究等，而这些将有待进一步的努力。

参 考 文 献

1 杨芙清，梅宏，吕建，金芝. 浅论软件技术发展. 电子学报，2002，30(12A)：1901～1906
2 Sun Microsystems. Jini Architectural Overview. White Paper, 1999
3 Microsoft corporation. Universal Plug and Play Device Architecture Version 1.0. June 2000
4 IETF SVRLOC Working Group. Service Location Protocol. http://www.srvloc.org
5 Object Management Group. CORBA, OMG Website. 2000. http://www.omg.org
6 Microsoft Corporation. Microsoft COM homepage. 2000. http://www.microsoft.com/com
7 Sun Microsytems. Java 2 Platform Enterprise Edition Specification. http://java.sun.com/j2ee.
8 Roman M, Kon F, Campbell R. Reflective Middleware: From Your Desk to Your Hand. IEEE Distributed Systems Online, 2001, 2(5)
9 Grace P, Blair G S, Samuel S. ReMMoC: A Reflective Middleware to Support Mobile Client Interoperability. In: Proc. of Intl. Symposium on Distributed Objects and Applications(DOA), Catania, Sicily, Italy, Nov. 2003
10 Szyperski C, et al. Component Software: Beyond Object-Oriented Programming. Pearson Education Limited, 2003
11 吕建，张鸣，廖宇，陶先平. 基于移动 Agent 技术的构件软件框架研究. 软件学报，2000，11(8)：1018～1023
12 陶先平，等. Mogent 系统的通信机制[J]. 软件学报，2000，11(8)：1060～1065
13 Smith BC. Reflection and Semantics in a Procedural Language: [Technical Report 272]. MIT Laboratory ofComputer Science, 1982
14 Hursch W, Lopes C V. Separation of Concerns: [Technical Report]. College of Computer Science, Northeastern University, Boston, MA USA, 1995