移动 Agent 系统的服务动态部署研究

孙海鸣,王志坚
(河海大学 计算机及信息工程学院,江苏 南京 210098)

摘 要:通过使用移动 Agent,分布式应用可以像一组可移动的松耦合的组件那样执行它们的功能。因此可以做出模块化并且可扩展的设计。但是,目前的大多数移动 Agent 平台的扩展性并不好,平台配置之后往往就很难再改变。为此,在动态配置的支持下为移动 Agent 设计了一个基于组件的框架,使得组件可以在运行时被增加、移动和改动,为构建一个高适应性的移动 Agent 平台提供了可能。

关键词:移动 Agents;动态部署;Java;服务
中图分类号:TP311.5 **文献标识码**:A **文章编号**:1673-629X(2006)06-0171-03

Dynamic Deployment of Services on Mobile Agent Systems

SUN Hai-ming, WANG Zhi-jian
(Computer and Information Engineering College, Hohai University, Nanjing 210098, China)

Abstract: By using mobile agent, distributed applications can implement part of their functionality as a set of loosely-coupled components, which are able to migrate. This allows the creation of very modular and extensible designs. Nevertheless, most mobile agent platforms are not extensible themselves. The platform does not accommodate changes after being deployed. In this paper, describe a component-based framework for mobile agents with support for dynamic reconfiguration, so that components can be added, removed and reconfigured at run time, with minimal disruption to the application, and make it possible to design a flexible mobile agent platform.

Key words: mobile agents; dynamic deployment; Java; services

0 引 言

近几年,基于组件的中间件系统不断受到大家的关注,主要是因为它们比原来的集成系统具有更大的灵活性。组件系统的改进和维护变得简单,而且具有良好的适应性。移动 Agent 有着很多和组件相同的优势。移动 Agent 具有良好的智能性和适应性,并且可以很好地被复用。它的另一个优势是可以在分布式系统的主机之间移动[1]。就像很多专家指出的那样,这种能力将使得未来的中间件能够比现在拥有更大的灵活性。一个基于移动 Agent 的分布式系统可以在几乎不需要中断应用的情况下进行动态修改,只需要结束一些 Agents,然后向等待更新的节点派遣新的 Agents[2]。

但是,大多数移动 Agent 系统中使用的中间件的灵活性停留在集成式系统的水平。这么做有很多坏处。一方面,这些平台一旦部署就很难再重新配置;另一方面,这些系统的可扩展性很差。往往使用静态 Agents 去扩展平台,提供系统级的服务。但是这实际上是对移动 Agent 概念的滥用,阻碍了移动 Agent 的广泛应用。为了摆脱这种限制,出现了一种用 Java 实现的为移动 Agent 设计的基于组件的中间件(即:M&M 框架)[3]。它是由一些 JavaBeans 组件构成的,其中有一个基础的移动组件支持一些核心功能,包括可移动性、安全性和可扩展性。这个组件可以像其他 JavaBeans 组件那样很容易地组合到任何应用,这使得创建移动 Agent 的应用变得容易。设计者可以通过往扩展层中添加服务来增加它的功能。这些服务在添加之后,对扩展层的所有客户端,包括应用、移动 Agent 和其他服务都是可用的。

尽管这个模型已经被用来成功地执行和部署一些服务[4],像 Agent 跟踪、inter-agent 通信,但是它仍然局限于支持静态部署。而且,删除或者更新服务将不得不将系统停止和重启,给客户带来不便。故应改进这个模型,增加对动态配置的支持。

文中主要介绍在 M&M 框架中,怎样动态配置服务。使得服务可以被增加、删除和更新,而几乎不影响应用或者移动 Agents 的运行。并且描述了怎样通过 RMI 或者移动 Agent 来远程重新配置。

1 支持动态服务

1.1 要 求

支持动态配置的系统一般来说需要做到下面三点:

1)说明管理做了哪些改动;

收稿日期:2005-09-19
作者简介:孙海鸣(1981-),男,江苏盐城人,研究方向为软件复用;王志坚,教授,博士生导师,研究方向为软件复用、计算机网络、形式规约等。

2)系统应该在重新配置之后能够正常工作;

3)应用在重新配置期间不会中断。

这里将主要讨论后两点,即:支持在不中断系统运行的情况下更新程序,并且保持程序的连贯性。

1.2 Java和动态部署

Java平台有一些十分有用的特性用于实现动态部署,比如动态类加载和反射。

Java类加载器允许在运行时利用多种不同资源创建应用程序,并提供命名空间机制,使同一资源的多个版本在单个虚拟机上共存。Java类加载器体系结构提供在运行时组装应用程序所需的服务。使用类加载器,可以查找和验证类、资源和配置设置。类加载器服务不仅用于加载,还可以处理实际应用程序部署问题。例如,可以从不可靠的网络位置安全地加载资源;可在不关闭应用程序的情况下重新加载组建的更改版本,甚至允许一个组件的多个版本同时共存。

反射是Java的一个重要特征。使用反射,可动态显示Java类的特性:基类、超接口、方法签名和字段类型。还可用反射动态实例化对象、调用方法以及转换字段。这些特性对于支持动态部署是十分重要的,它使得一个框架可以在运行时和那些编译时无法知道的组件交互。

1.3 剖析一个服务

站在项目的角度来看,一个服务可以被看作一个JavaBeans组件,它必须包含一个服务描述者、一个服务提供者、一个服务接口(如图1所示)。

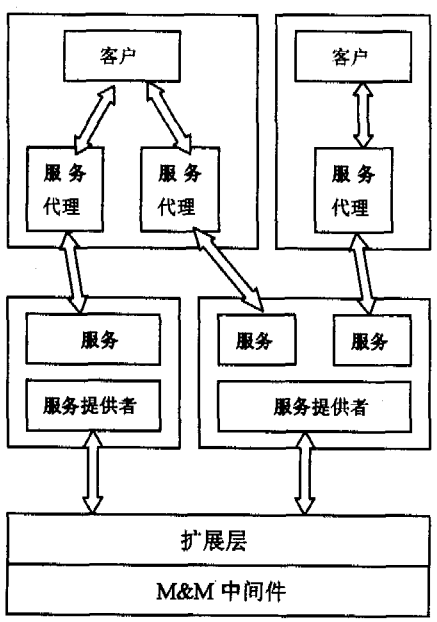

图1 动态服务结构图

服务描述者用来描述服务、定义服务。它包括名字、版本和接口名。如果两个服务有相同的名字和接口,那么他们是兼容的。这使得服务可以更新。

服务提供者被中间件用来和服务交互。它提供方法去创建新的实例、改变配置和获取服务描述。

服务接口说明了客户如何和接口交互,并且为接口定义了功能。每个服务必须为客户提供一个明确的接口。同时,服务必须实现这个接口。

1.4 执 行

实现运行时的加载和卸载服务,有必要对每一个服务使用不同的类加载器加载。这使得每个服务成为独立的个体。甚至可以并行执行一个服务的不同版本。当一个服务的版本不是向后兼容时,这是必需的。

但是,仅仅通过使用单独的类加载器对于服务的卸载和替换来说是不够的,因为一个服务和它的类被卸载时,这个服务的类加载器也必须被卸载。一个用户定义的类加载器在虚拟机中也是作为一个对象存在的,和其他对象遵循同样的规则:当一个对象已不关联任何引用时,就会被垃圾收集器回收。类的各个实例维护一个到其Class对象的引用,通过getClass()方法访问;另外,Class的各个实例维护一个到其ClassLoader的引用,通过getClassLoader()方法访问。反过来,类加载器加载的任何类的单个实例将在内存中保存类加载器,及类加载器加载的各个类。因此,服务必须通过特定的方式加载和引用,来保证能够清除和中间件、客户或者应用相关的类加载器、类以及类的实例之间的关联。

处理到一个对象的所有引用并不复杂,只需要跟踪所有引用(后面将解释怎样通过代理实现)并且在卸载的时候把它们设为null。处理到类的引用时就相对复杂一点。因为在装载的时候,一个类只是对它引用的其他类包含一个象征的表示。然后,在这个类第一次引用其他类之前,系统才真实调用这些类。

另外,如果一个服务外部的类获得了服务定义的类的引用,那将会产生问题。因为如果出现这样的情况,那么只有当这个外部的类卸载之后,才能卸载服务。这往往意味着要中断整个系统。因此,这样的情况要避免。

可以通过在服务中制定两个接口,分别针对中间件和客户端,来避免对服务的直接调用。在与中间件的交互中通过多态引用避免直接调用,而在与客户端的交互中使用代理。

中间件和服务的交互类似于applets和servlets,applets和servlets都必须继承一个抽象类并且这个抽象类将用于所有的交互。服务必须继承一个叫ServiceProvider的抽象类。当加载这个服务时,中间件可以得到这个抽象类,并且在和这个服务的所有交互中使用它,不过通常是用对这个类的一个引用。多态性使得对服务的调用可以通过使用ServiceProvider的引用。通过这种方法,中间件就避免了直接调用服务。

服务和客户端之间的交互就不能使用上面的方法。因为使用服务的客户是知道客户和服务之间的接口的,否则,他们就不能使用这些服务。但是中间件是不知道这些接口的,不然,就不能被称为动态部署了。客户端必须通过某种方式调用服务中的接口,而服务的代码中定义的这

个接口必然是用加载该服务的类加载器加载的。如果客户调用这个接口,那么就意味着客户对服务有一个直接的调用。另外,如果当一个服务还没有重新部署好时,一个客户要调用这个服务,在这种情况下,客户应该被通知服务没有准备好,并且等待。但是,如果客户是直接调用服务,那么一个 ClassNotFoundException 异常就会抛出,并且客户端也会终止执行。上述问题都要求客户端有服务接口的定义,这样,使得这个接口是被客户的类加载器定义的。这意味着服务的接口将会有 2 个相互矛盾的定义,一个是用客户类加载器加载的,另一个是用服务的类加载器加载的。这就又造成另一个问题,当被请求创建一个实例时,服务创建一个用服务的类加载器定义的对象。这个接口和通过客户类加载器加载的那个接口是不兼容的,客户不能直接调用。这个问题可以通过使用动态代理 API 解决。动态代理通过反射来调用服务对象。这避免了不兼容和直接调用。

1.5 更新服务

加载、卸载和创建服务的新实例都很简单,故在此不加以说明。但是更新一个服务是一个复杂的过程,下面将详细说明。

更新一个服务,并不是简单地先卸载旧的版本,然后加载一个新的版本。因为,这样对客户来说不是透明的,会造成对旧的服务实例以及所有相关状态的访问的突然中断。这样,客户端将会调用失败,或者将不得不请求服务的新实例,中断和前一个版本的所有连接。

要使系统不中断运行,就应该让更新对客户端透明。这是一个难题。要注意三个要点:第一,旧版本的对象的状态必须能够被传递给新版本的对象,就算前后两个类的构造不一样,也必须能够相适应;第二,旧版本的所有静态字段在新版本中必须保留;最后一点,必须保证客户没有调用将要被更新的类的方法。

前两点都是服务设计者的责任,他们了解服务的行为。举例来说,如果新旧版本的构造不一样,或者如果服务已经有了活动的线程或已锁,中间件就无法更新。因此,如果一个服务将会被更新,那么在设计时,就要有规范的方法来保存该服务及其所有运行实例的状态。

第三个问题可以通过对每一个服务使用读-写锁来解决。一个服务的所有代理必须取得 read lock 才能执行服务的方法。当开始更新时,ServiceManager 获得一个 write lock,这样,所有的客户都无法连接服务。之所以有效是因为客户连接服务的惟一方法就是通过代理,所以可以认为当 ServiceManager 拥有读权限时,没有客户在调用服务。

1.6 服务管理器

上文描述了中间件是怎样支持和管理服务的。现在还有必要说明:中间件是怎样发现服务的,服务是怎样管理和配置的,安全是怎样实现的,服务是怎样和外部应用交互的。由于篇幅限制,仅简要说明比较重要的方面。

中间件可以提供一个 ServiceManager 接口,使得应用可以控制服务。这个接口可以允许服务的安装、删除、开始和停止执行、部署和例示,并且被井井有条地管理,保证只有授权的客户端才能访问它。

这个 ServiceManager 接口有可能就在应用的本机上,也有可能是远程的。中间件可以通过一个 RMI 接口调用 ServiceManager。同样,一个服务也可以这样调用 ServiceManager 接口。这样,Services 就可以通过 RMI 或者移动 Agent 来远程配置和部署。

另外一个要注意的地方是服务的配置。大多数服务都需要一个在运行期可改变的配置。如果在设计阶段,中间件就能知道服务,那么服务提供者就可以定义标准的 JavaBeans 属性,并且利用它们改变配置。但是,既然是动态服务,那么在编译时,中间件是不知道服务的具体细节的。有两种机制可以用来解决这个问题:第一种,使用反射来获取服务提供者提供的服务的属性,ServiceManager 负责管理这些属性,允许应用来查询和修改它们;第二种,应用可以向服务广播或者传递消息,ServiceManager 再把这些消息传递给服务提供者,由他们据此来使得服务和应用相适应。

2 相关工作

国外已经有一些移动 Agent 的系统支持静态扩展,比如 MOA 和 Gypsy[5]。但是,它们目前都不支持动态配置。

支持动态部署的可扩展构架是最近几年的研究热点。比如 dynamicTAO 还有国内的 PKUAS。dynamicTAO 是一个反射 CORBA ORB,在 TAO ORB 的基础上构建。TAO 是一个支持静态部署的可扩展 ORB。Dynamic TAO 扩展了 TAO,可以实现动态部署。PKUAS 是北京大学软件研究所提出的一种反射式软件中间件。

文中提出的构架和这些系统有一些相似的地方,但是它们针对的是不同的问题。这些系统致力于构建一个通用的结构来支持动态部署,并且在不同的地方应用。而文中主要针对 Agent 系统,致力于构建一个可扩展的中间件来支持移动 Agent 应用。

3 总　结

描述了如何在移动 Agent 系统中进行动态重配置。在中间件中加入了一个扩展层来支持运行时增加或者更新服务,以便为 Agents 提供新的功能。这使得构建一个高适应性的移动 Agent 平台成为可能。文中只是针对一种移动 Agent 系统进行设计的,但是它也可以推广到其他移动 Agent 系统中去。

参考文献:

[1] 郭忠文,汪　洽,冯业伟.基于移动中间件的分布式计算研究[J].计算机应用研究,2003(3):60-62.

(下转第 176 页)

2.2.3 变异算子

在遗传算法计算初期各种群中染色体的值分散性很大,但在寻优过程中很快就能将搜索范围缩小在一个很小的范围内,这时就希望变异操作尽量在小范围内进行,但为了保持种群多样性和跳出局部最优解,也要有一些大范围的变异操作。文中采用了非一致性变异,在进行变异操作时,以 0.3 的概率进行普通变异的同时还以 0.7 的概率进行变异范围随着迭代次数的增加而缩小的小范围变异。

取一个 100 以内的随机数 R,如果 $R < 30$ 则进行普通变异,首先随机产生两个变异位 i,j(产生方法及对应关系和交叉算子中使用的相同),然后再产生两个 10 以内的随机数 m,n,分别替换染色体中经纬度对应位置的数值。

如果 $R \geqslant 30$ 则进行小范围变异,这时变异范围不再是小数点前第一位到小数点后第三位,而是由当前进化代数决定的。如果当前进化代数大于最大代数的 2/5,则只在小数点后第一、二、三位上进行变异;如果当前进化代数大于最大代数的 3/5,则只在小数点后第二位和第三位上进行变异;如果当前进化代数大于最大代数的 4/5,则只在小数点后第三位上进行变异。

例如 $C_1 = (117.234, 24.568)$,最大迭代次数 1000,在第 350 次迭代时进行变异操作。首先产生随机数 R:

如果 $R < 30$ 进行普通变异,获得 $i = 1, j = 2, m = 5, n = 4 (i,j \in [0,3])$,则变异之后的结果 $C_1' = (117.534, 24.548)$;

如果 $R \geqslant 30$ 则进行小范围变异,由于 $650 > 1000 * 3/5$,获得 $i = 2, j = 3, m = 5, n = 4 (i,j \in [2,3])$,则变异之后的结果为 $C_1' = (117.254, 24.564)$。

3 实验结果

实验结果如图 1 所示,左侧多边形为红方阵地边界,右侧多边形为蓝方阵地边界,蓝色圆点代表蓝方观察所位置,5 个红色小旗分别代表红方指挥所位置。基本指挥所位于阵地的中部,距敌方阵地前沿约 8km;预备指挥所在基本指挥所附近约 1km 处;前进指挥距离蓝方阵地前沿约 6km,距离基本指挥所约 2km;后勤和装备指挥所靠近红方阵地后方,分别距离基本指挥所约 2.5km 和 1.7km。可以看出各个指挥所所在位置除了满足其基本的地形条件外,相互之间也保持了一定的协同度。

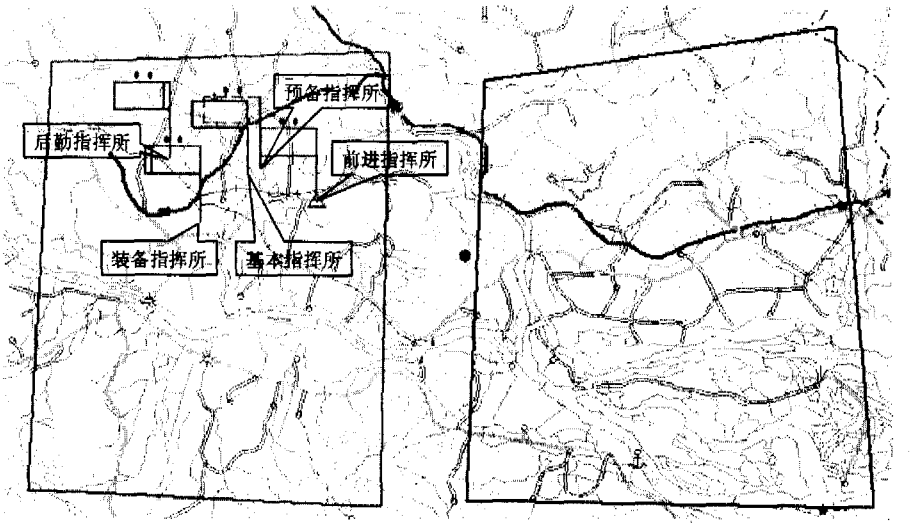

图 1 指挥所配置结果

4 结束语

地域选取是基于 GIS 的辅助决策系统的关键技术之一,多目标的地域选取是地域选取中比较复杂的问题。文中以指挥所配置为例将改进的协同进化遗传算法应用于多目标地域选取,不但成功地解决了问题,而且取得了令人满意的结果。但是协同进化遗传算法计算复杂性较高,如何减小计算复杂性,还需要进一步研究。

参考文献:

[1] 毕思飞,吴裕树,郑建军. 一种基于 GIS 和遗传算法的地域选取方法[J]. 微机发展,2004,14(6):13-15.

[2] 黎 夏,叶嘉安. 遗传算法和 GIS 结合进行空间优化决策[J]. 地理学报,2004,59(5):745-753.

[3] 陈海涛,赵 有,吴启权. Gause 竞争型协同进化算法在 FNN 中的应用[J]. 计算机工程与应用,2004(34):85-91.

[4] 张运凯,王方伟,张玉清,等. 协同进化遗传算法及其应用[J]. 计算机工程,2004,30(15):38-43.

[5] 总参谋部. 军事地形学(第 2 版)[M]. 北京:解放军出版社,2000.328-330.

(上接第 173 页)

[2] Jing J, Helal A S, Elmagarmid A. Client-Server Computing in Mobile Environments[J]. ACM Computing Surveys, 1999, 31(2):117-157.

[3] Halloway S D. Component Development for the java Platform [M]. 北京:清华大学出版社,2004.

[4] Bellavista P, Corradi A, Stefanelli C. Mobile Agent Middleware for Mobile Computing[J]. IEEE, 2001, 34(3):73-81.

[5] Jazayeri M, Lugmayr W. Gypsy: A component based mobile agent system[A]. 8th Euromicro Workshop on Parallel and Distributed Processing[C]. Rhodos, Greece: [s.n.], 2000.