一种支持自动部署的脚本语言的设计与实现*

陶　聪, 李　巍, 李云春
(北京航空航天大学 计算机学院, 北京 100083)

摘　要: 提出一种主计算平台下的应用编辑脚本语言 ACSPT, 基于此脚本语言, 可构造基于自主计算平台的大规模分布式网络应用。

关键词: 自主计算; 自动部署; 脚本语言
中图法分类号: TP311　　文献标识码: A　　文章编号: 1001-3695(2005)11-0031-03

Design and Implementation of Scripting Language for Self-deployment

TAO Cong, LI Wei, LI Yun-chun
(School of Computer Science & Technology, Beihang University, Beijing 100083, *China)*

Abstract: A script language, ACSPT, which is used to build application under the autonomic-computing platform, is presented in this paper. Based on this language, application of large-scaled distributed work under the autonomic-computing platform can be built.

Key words: Autonomic Computing; Self-deployment; Script Language

随着 Internet 网络技术和基于 Internet 的应用和服务技术的快速发展, 基于 Internet 的分布式网络应用技术越来越庞大、复杂, 对于管理人员的需求也在不断地增加, 管理人员的缺乏成为制约大规模网络应用技术发展主要问题。在这种背景下, 由 IBM 公司提出自主计算的概念, 期望解决这一问题。自主计算思想来源于人体的植物神经系统, 植物神经系统可在无大脑意识控制的条件下自动地管理人体局部功能, 目的在于实现计算机系统的自动管理, 减少人为干预, 实现系统的自动部署、自动配置、自动修复、自动优化、自我保护。

1　应用背景和存在的问题

对于自主计算平台的研究, IBM 公司支持的有 AUTOMATE 项目, 亚利桑那州大学的 AUTONOMIA 项目; 中国科学院计算技术研究所智能信息处理重点实验室对自主计算中智能主体和机器学习有机整合也进行了深入的研究。这些工作主要集中于操作系统之上应用程序之下的自主计算平台上, 自主计算平台提供了一些 API(应用程序接口), 但使用这些接口编程的方式非常复杂, 这要求开发人员了解自主计算平台的内部原理, 要求开发人员远程调用模式等, 这些概念的复杂性限制了它们的实际应用。

用于组件或服务组合应用的主要技术有 IBM 公司的万维网服务语言(Web Service Flow Language, WSFL), 微软公司的 XLANG, 以及 IBM 公司和微软公司联合推出的业务流程执行语言(Business Process Execution Language, BPEL)[1]。万维网服务能够以松散方式把不同的服务集成起来, 形成一个新的服务, 完成特定的功能[2]。通过这些语言提供的结构化活动, 可以将组件或服务提供的原活动组成更复杂的算法。在国内, 网格服务标记语言(GSML)是中国科学院计算所提出的面向网格环境下最终用户的应用开发语言, 能够把多个服务组合起来, 构成一个复合服务。

目前, 自主计算平台的使用只针对专业技术人员, 缺少一种有效、好用的平台编程和使用界面工具; 基于 WSDL 的分布式组件协调语言, 只能被动地调用现有的 Web 服务, 不能主动地控制 Web 服务的生命周期, 不能动态地部署组件, 不能利用自主计算平台的特性, 不支持应用的自我配置, 不适合自主计算环境下的应用组件或服务的集成。

如何简单有效地利用自主计算平台和部署在上面的组件快速组成应用, 如何协调组件之间的调用关系, 如何利用自主计算平台特性使应用具有自动配置功能是本文所要关注的。

2　自主计算平台的组织模型

自主计算运行平台的组织模型如图 1 所示。系统主要由以下部分组成: 应用管理编辑器、策略执行引擎、计算节点(自主计算组件运行容器和资源监视器)、组件库和资源库。平台中的组件, 是可部署的计算单元, 部署后成为一个服务实例。

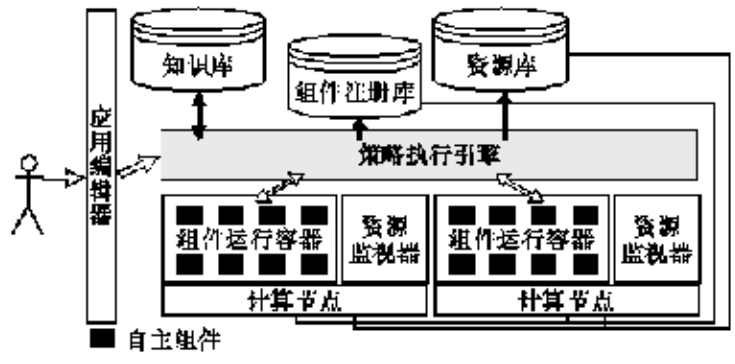

图 1　自主计算运行平台组织结构

应用编辑器是系统的用户接口, 编辑器一方面从组件库中读取注册的组件信息并显示给用户, 以便用户调用; 另一方面, 用户可以在编辑器制定组件的运行策略, 编写组件运行脚本形成自主计算的应用, 编辑器将用户编写好的应用组织为脚本文件, 提交给策略执行引擎。组件库中记录在系统中已经注册的

收稿日期: 2004-10-31; 修返日期: 2005-01-20
基金项目: 国家“863”计划资助项目(2003AA115230)

组件详细信息;资源库中存放系统管理的所有计算节点的计算资源,如磁盘、网络信息等;组件库和资源库中的信息是动态刷新的。策略执行引擎是系统执行的调度中心,负责组织各个计算节点下载、部署、迁移组件。计算节点上驻留两种代理,即组件运行容器和资源监视器。组件运行容器是组件真正的运行环境;资源监视器监视计算节点中的计算资源,同时产生各种事件,通知策略执行引擎进行策略调度。

平台使用关系型数据库作为自动配置知识库的数据模型。数据库中主要包括六张表:主机表、组件表、组件实例表、条件表、动作表和策略表。策略表中记录着条件发生后需要采取的动作,一个条件发生以后,可以采取多种策略来重新配置系统,是一对多的关系。策略表中还记录每一种策略的权,采取这种策略以后,资源监视器监视计算节点运行情况,并以远程事件的形式报告策略引擎,以反馈的形式更新策略的权值,保证策略的动态优化,如图 2 所示。

3 基本语法结构

针对自主计算平台的特征,提出适合编写自主计算应用的脚本语言——ACSPT。ACSPT 以现有的脚本语言为基础,语法比较简单,容易理解和学习,面向应用的最终开发者;同时扩展语法成分,紧密结合自主计算平台的特性,使用该脚本编写应用,可以容易地实现自动部署和自动配置。

用户使用 ACSPT 编写一个 ACSPT 页面并部署到自主平台上,然后就可以通过一个浏览器来使用这个页面,从而获得自主平台上的服务。在运行时,ACSPT 页面的计算请求从客户端浏览器传到自主平台的一个服务器上,后者解释执行所需的服务后将结果传回浏览器,如图 3 所示。

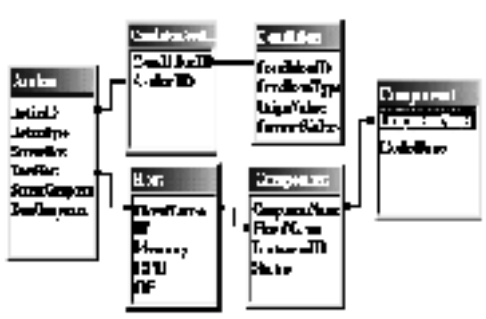

图 2 知识库构成

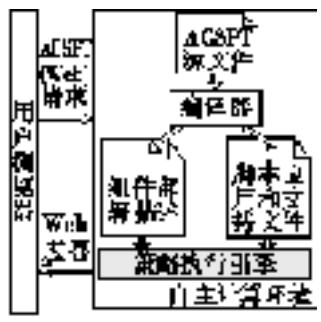

图 3 ACSPT 执行过程

一个 ACSPT 脚本页面由编译预处理、保留部分、静态输出和动态输出三部分组成。其 BNF 语法如下:

< acspt > : = < include > | < reserved > | < static > | < dynamic > } *

其中 < include > 语法成分类似于 C 语言中的编译预处理指令“include”,可以包含其他文件,以适合应用任务分解和多人协调开发。< static > 为页面静态显示的内容,能兼容 HTML 格式字符。< reserved > 部分为自主计算平台其他子系统保留,例如认证信息和访问控制列表等安全控制信息,以实现自主计算平台中基于角色的访问控制。ACSPT 解释器将忽略该语法成分。< dynamic > 为自主组件调用脚本,是 ACSPT 的核心。< dynamic > 由组件定义和组件调用两部分组成。其语法如下:

< dynamic > : = { < component > | < script > } *
< component > : = { < service > } *
< script > : = { < jsx > } *

语法成分 < component > 为组件定义部分,其中包含了本次计算所需要的所有服务组件的声明,这类似于高级语言的变量声明。为了支持自主计算的平台特性,用户服务组件的属性和运行时策略,语法如下:

< service > : = { < codebase > } * { < shared > } 1 0 { < sla > } 1 0 { < group > } * { < description > } 1 0
{ < events > } * { < depends > } *

< codebase > 为该组件包的物理位置,自主计算平台通过 < codebase > 来下载和部署该组件,缺省时平台在组件库中查找已经注册的组件,也可以指定多个 < codebase >,平台将依次查找组件包,直到找到为止。如果组件定义中含有 < shared > 成分,则在自主计算社区中共享一个组件实例;否则,自主计算平台部署一个新的组件实例。< group > 指定组件运行时所在的组,< description > 对组件进行注释性的描述。< depends > 描述了该组件对其他组件的依赖关系。

在 < sla > 中可以为组件指定运行时的服务水平(service level agreement)。其语法如下:

< sla > : = { < platform > } 1 0 { < runtime > } 1 0
< platform > : = { < os > } 1 0 { < jrever > } 1 0
< runtime > : = { < cpu > } 1 0 { < memory > } 1 0 { < netband > } 1 0

以下的代码是一个组件运行时需求的例子。该实例指定组件需要在 Windows 2000 操作系统上运行,并且需要 1.41 版本以上的 Java 运行环境支持。在组件运行时,CPU 主频为 1.7GHz以上,利用率不能超过 0.8;内存容量不能低于 265MB,利用率不能超过 0.8;网络带宽不能低于 300kbps。

```
< sla >
  < platform >
    < os value = " win2k" >
    < jrever value = " 1.41" >
  < / platform >
  < runtime >
    < cpu value = " 1.7g" shrold = " 0.8" >
    < memory value = " 256m" shrold = " 0.8" >
    < netband value = " 300k" >
  < runtime/ >
< sla/ >
```

4 组件协调语法成分的实现

在自主计算平台内部,系统已经提供了一些基础的服务组件,如时间服务、监视服务等。而专业领域计算中需要的服务组件,可以由其他自主计算系统程序员开发和部署。在 ACSPT 法成分 < jsx > 的主要功能是调用和协调自主平台提供的服务组件。相对于 Java 这样的系统编程语言来说,语法成分 < jsx > 很少用于实现复杂的算法和数据结构,是一种具有更高抽象层次的简单易用的编程语言。

语法成分 < jsx > 没有重新定义一套自己的词法和语法规则,而是扩展了 Netscape 的脚本语言 JavaScript。JavaScript 是一种脚本编写语言,它采用小程序段的方式实现编程,像其他脚本语言一样,JavaScript 同样也是一种解释性语言,它提供了一个简易的开发过程。基本结构形式与 C, C ++, VB, Delphi 十分类似。但它不像这些语言需要先编译,而是在程序运行过程中被逐行地解释。作为一种脚本言语,JavaScript 本身就有较强的表达能力,它是一种基于对象和事件驱动的编程语言,这是非常适合控制自主平台的服务组件。但是要成为自主计算平台的应用编辑语言,JavaScript 还要作如下扩展:①对 Java 原生类(Native Class) 支持;②支持 Java 的远程对象;③提供远程事件回调支持;④添加对组件生命周期管理的内置函数。

在自主计算平台下,服务组件都是由 Java 语言写成。ACSPT 语言作为组件之间的协调语言,必将涉及到大量的 Java 原

生类,调用组件时参数的传入,调用完成后参数的返回等等。原有的 JavaScript 语言未提供对 Java 类库的调用功能,< jsx > 中扩展了 JavaScript 的语法,添加了关键字 Native,用户可以方便地构造 Java 类。例如,var str = native java. lang. String(“Hello ”) 。ACSPT 脚本解释器集成了开源项目 FESI 解释器的功能,可以方便地使用 Java 原生类。

与其他分布式系统类似,自主计算平台中组件调用也使用“桩 - 代理 ”调用模式。但是使用“桩 - 代理 ”调用模式编程,无疑提高了自主计算平台下应用开发和维护的难度。ACSPT 提供了支持 Java 远程对象的能力: ACSPT 脚本解释器在解释时,会识别脚本中的远程对象,通过组件描述,在脚本中插入远程对象接口的声明,并生成本地的桩。这样,程序员使用远程对象和使用本地对象语法是一致的。

在自主计算平台下,对象之间的通信采用异步的方式。当发生某些预先定义的事件时,自主组件可以回调一段与这些事件相应的程序。但是直接调用平台提供的应用程序接口(API) 比较复杂,需要经过请求、注册、回调等一系列的过程。在 ACSPT 语法中,用户使用类似 c1. onEvent = c2. method(c1, c2 皆为组件实例) 的语句就可以实现对象之间的异步通信和协调。ACSPT 脚本解释器在解释执行时,将查找组件 c1 的事件 onEvent,如果是远程事件,则将 c2. method 注册为该事件的回调方法,一旦产生远程事件后,策略执行引擎通知组件 c2 所在的组件运行容器,调用组件 c2 的 method 方法。

虽然自主计算平台能够自主地管理组件的生存期,程序员也可以显式地调用组件生命周期管理内置函数来完成对自主组件的管理。组件模型的生存期状态图如图 4 所示。程序员可以通过调用 initialize( ) , start( ) , stop( ) , destroy( ) 等内置函数来管理组件生命周期。ACSPT 脚本编译器处理以上函数时,将在脚本中插入查找和定位组件的代码,从而以远程事件的形式通知组件所在的容器,对组件进行相应生命周期的管理。

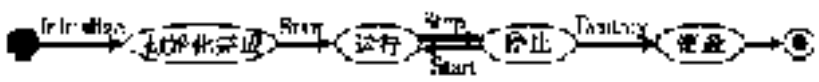

图 4　组件生命周期

5　应用实例

以下的代码是一个基于自主计算平台的网络分布式存储应用实例。该应用能够实现数据的分段存储,保证了存储的地址透明性,并具有一定的负载均衡和自动配置的功能。脚本程序员不需要关心如何实现存储的地址透明性和负载均衡,这些功能由系统运行时自动完成。该应用需要两个核心组件,即存储组件 SaveAgent 和目录管理组件 IndexAgent。前者主要负责将数据保存到所在的计算节点上,后者主要负责文件目录的管理,实现文件的分段存储。这两个组件由系统程序员编写。在本实例中,脚本语言只是调用接口。

在脚本开始声明组件的运行要求的服务水平。为了保证存储数据的空间,要求 SaveAgent 组件所在的计算节点磁盘空间大于 300MB,为了提高文件目录的检索速度,要求 IndexAgent 组件运行时 CPU 利用率不能超过 70%。如果在 SaveAgent 运行时,计算节点的磁盘空间小于 300MB,计算节点的资源监视器将信息以远程事件的形式通知本地,从而触发 DiskFailure 函数的执行。

```
< service >
  < name > < /name >
  < codebase > 192. 168. 2. 23/ SaveAgent. jar < /codebase >
  < sla >
    < runtime >
      < storage shrold = "300m" >
    < runtime / >
  < sla / >
  < name > IndexAgent < /name >
  < codebase > 192. 168. 2. 35/ IndexAgent. jar < / codebase >
  < sla >
    < runtime >
      < cpu value = "2. 0g" shrold = "0. 7" >
    < runtime / >
  < sla / >
< /service >
```

编写初始化组件的代码,策略引擎将查找合适的计算节点,部署组件并完成初始化的工作。

```
< jsx >
  SaveAgent[ 3] sas;
  for( int i = 0; i < = 2; i + + ) {
    sas[ i] = rnew SaveAgent;
    init( sas[ i] ) ;
    start( sas[ i] ) ;
  }
    IndexAgent ia = rnew IndexAgent;
< /jsx >
```

编写代码存储事件结束的时候

```
function AfterStore( ) {
    IndexAgent. SaveIndex( file_name, index, url) ;
    Destroy( this) ;
}
```

编写代码存储处理失效的事件。SaveAgent 驻留在远端的机器上,有多种失效的可能,这些信息以远程事件的形式返回,从而触发以下代码的执行-重新查找合适的计算节点进行存储。

```
function DiskFailure( index) {
    sas_later = rnew SaveAgent( ) ;
    init( sas_later) ;
    start( sas_later) ;
    sas_new. store( ) ;
}
```

6　结论

本文所讨论的构件模型及其描述语言具有两个特点: ①适用自主计算平台的应用构造,即能够动态地管理组件的生命周期,利用平台的策略库,可以自动部署应用; ②易用性,即以简单脚本语言为基础,提供基于对象的编程方法,统一远程对象和本地对象的调用方式。

参考文献:

[1]　eb服务的业务流程执行语言[EB/OL] . http: / /www-900. ibm. com, 2003.
[2] Automatic Computing, Creating Self-managing Computing Systems [EB/OL] . http: / /www-306. ibm. com/, 2003.
[3] Salim Hariri, Lizhi Xue, Huoping Chen, *et al*. AUTONOMIA: An Autonomic Computing Environment[C] . IEEE International Performance Computing and Communications Conference, 2003.
[4] 徐志伟,冯百明,李伟. 网格计算技术[M] . 北京: 电子工业出版社, 2004. 215-220.
[5] 付岩,白硕,李国杰. 一个集成了 COM 和 CORBA 的脚本语言[J] . 软件学报, 2001, 12(6) : 840-845.
[6] 严莉萍,鲍敢峰,尤晋元. 一种基于分布组件协调的脚本语言 Concerto[J] . 上海交通大学学报, 2001, 35(2) : 188-191.

作者简介:

陶聪(1981-) ,男,四川南充人,硕士研究生,研究方向为软件理论、分布式网络; 李巍(1970-) ,女,北京人,副教授,硕士生导师,研究方向为网络测量、分布式网络; 李云春(1972-) ,男,陕西人,讲师,研究方向为网络测量、分布式网络。