分布式虚拟环境平台(NPSNET-V)的研究

薛晓明　敬万钧　刘锦德
(电子科技大学计算机学院　成都 610054)

摘　要　NPSNET-V 由美国海军研究生院开发,其目标是逐步发展形成一个支持互联网上分布式虚拟环境的框架。目前,NPSNET-V 是一个支持网络虚拟环境应用的、基于组件的、动态可扩展平台。本文论述了它的基本构思、组件框架、实体模型、典型架构、应用结构、设计启示等。

关键词　NPSNET-V,分布式虚拟环境,动态可扩展平台

Study on NPSNET-V, a Platform for Distributed Virtual Environments

XUE Xiao-Ming　JING Wan-Jun　LIU Jin-De
(College of Computer Science and Engineering, UESTC of China, Chengdu 610054)

Abstract　The NPSNET-V, developed by U. S. Naval PostGraduate School, is aimed at evolving to be the framework for supporting distributed virtual environments on the Internet. At present, it is in a form of a component-based, dynamically extensible platform for distributed virtual environment applications. This paper presents its essential idea, component framework, entity model, typical architecture, application structure, and valuable guidance.

Keywords　NPSNET-V, Distributed virtual environments, Dynamically extensible platform

1　NPSNET-V 基本构思

NPSNET-V(Naval Postgraduate School Network 第五版的简称)由美国海军研究生院开发,其目标是逐步发展形成一个支持互联网上分布式虚拟环境的框架,它能在互联网上动态地扩展且规模上可缩放[1,6]。目前,NPSNET-V 是一个支持网络虚拟环境应用的、基于组件的、动态可扩展平台,它支持客户端应用、服务器端应用、对等应用及单独的应用。

如同其它虚拟环境工具包一样,NPSNET-V 包含完成图形绘制、用户输入、网络及相关服务的模块。与传统工具包不同的是:传统工具包在编译完成之后一切就被固定了,或是可通过 plug-in 界面实现有限的动态扩展;而 NPSNET-V 允许应用在运行时扩展或升级它的任何功能。NPSNET-V 的动态可扩展体系结构实现了接近无限度的可重新配置。NPSNET-V 支持的虚拟环境中的每一个实体都是由独立的代码模块生成的;为了向虚拟环境中添加新实体,开发者需要编写新实体的代码模块,将这些模块上传至公用 Web 服务器上,并在虚拟环境中实例化这些模块。当用户访问虚拟环境时,他们所在的主机自动从 Web 服务器上下载相应于虚拟世界内容的代码模块,并将这些模块加入 NPSNET-V 的框架中;将模块激活(初始化)后,新下载的模块就成为客户端应用的一部分,它们所完成的功能包括绘制实体、计算物理力、传送和接收网络更新、处理鼠标和键盘输入等。当虚拟世界演变时或当用户进入虚拟世界的新的区域时,客户端应用升级现有的模块,加入新模块或卸下冗余的模块以反映虚拟环境的当前状态。除了一个不变的最小内核之外,NPSNET-V 架构的其余部分全部是可以加载和集成的[2,3]。

NPSNET-V 框架完全基于 Java 平台而建立[3]。该框架有一个简单的微内核,微内核提供的基本功能是支持其它组件的配置和串行化(其作用为加载组件和保存组件)。其它组件以一种类似文件系统的目录结构的形式,在微内核上形成一个组件层次结构(容器层次结构),如图 1 和图 3 所示。组件间的关系是一种松耦合关系。组件间的交互通过界面层实现。最基本的界面是属性(property);各组件所提供的服务也是属性。当一个组件要求某种服务时,先在自己所在的容器中寻找提供该服务的组件,若未找到则向上一级容器继续寻找,如此一级级向上寻找提供该服务的组件。

2　NPSNET-V 组件框架

NPSNET-V 提供了一个应用可插入组件或模块的框架,典型的应用可以包含 NPSNET-V 所提供的模块、应用开发者编写的模块或第三方编写的模块。应用开发者编写的模块一般都从 Module 继承,Module 是 NPSNET-V 的 kernel 包中的抽象基类。该 kernel 包中还包含 ModuleContainer 和 Kernel 类。正如文件系统的层次结构一样,ModuleContainer 的子类(简称容器)可以包含其它模块,类似于文件系统的目录的作用;kernel 也是一个容器,它的地位类似于文件系统的根目录。NPSNET-V 框架中加载的每个模块都有一个本地名称,用来区别于同一容器中的其它模块;同时每个模块有一个全局名称(路径),用来确定它在 NPSNET-V 框架的层次结构中的位置。容器和命名层次如图 1 所示,其中箭头代表容器关系、方框代表框架中的模块、每个模块有一个本地名称(见各方框的上一行名称)和一个全局名称(见各方框的下一行名称)。

薛晓明　博士生,主要研究方向:虚拟现实、开放式分布系统、计算机图学。**敬万钧**　教授,主要研究领域为虚拟现实、计算机图学、计算机动画。**刘锦德**　教授,博导,主要研究方向:开放式分布系统、中间件和安全性技术。

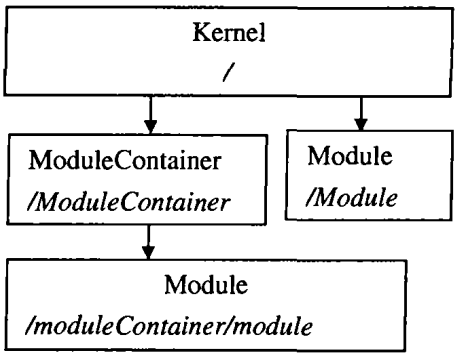

图 1 容器和命名层次

2.1 模块的生命周期[3]

NPSNET-V 框架中模块的生命周期如图 2 所示。每个模块的生命周期由该模块所在的容器控制。容器首先调用该模块的 init 方法，使模块从未初始化状态(Uninitialized)进入初始化状态(Initialized)。如果该模块实现了 startable 界面，当模块所在的容器调用模块的 start 方法时，该模块生成新的执行线程并进入启动(started)状态。当容器调用模块的 stop 方法来停止该模块的执行线程时，模块将返回初始化状态。当容器调用模块的 destroy 方法时，模块将进入销毁(Destroyed)状态。当模块被动态替换(指运行时用模块的更新的版本替换该模块)时，模块所在的容器通过调用新模块的 replace 方法和旧模块的 retire 方法实现这一替换，旧模块进入销毁状态。

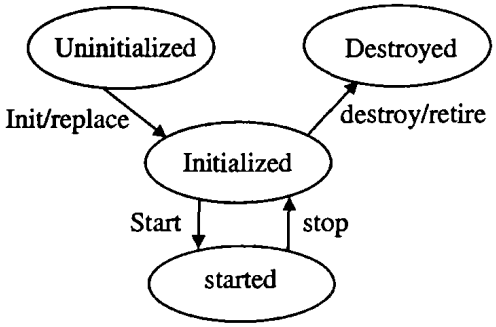

图 2 NPSNET-V 的模块的生命周期

2.2 界面层

NPSNET-V 的模块间通过 Java 方法调用互相通信；为此，各模块共享共同的基类或界面。这些基类和界面位于三个包中：kernel.jar，properties.jar，services.jar。在 kernel 包中的类和界面被系统中其它模块继承，但这些类和界面的方法局限于完成系统管理；NPSNET-V 为模块间的交互提供了可扩展的界面层，这些界面位于 properties.jar 和 services.jar，它们的内容代表了 NPSNET-V 平台的基本界面库。Property 界面从 PropertyBearer 继承；Service 界面从 ServiceProvider 继承；它们在 NPSNET-V 中有特殊的作用。例如，ModuleContainer 的 getPropertyBearers 方法使容器中的模块能得知在该容器中所有具有某个给定属性的模块。同样，ModuleContainer 的 getServiceProvider 方法返回某个给定服务的提供者；该方法首先在本地容器中寻找服务提供模块，若本地容器中未找到，它就在上一层容器中寻找，若未找到就继续向上查找。服务提供模块一般位于以微内核为根的层次结构中的某个容器内，它的作用是为所在的子层次结构中的模块提供服务。

2.3 模块的加载和保存

NPSNET-V 用两种类型的 XML 文件加载和保存模块：配置文件和原型文件。配置文件用于记录各模块的状态和层次关系；原型文件记录模块本身的情况(包括模块的配置及模块的类和依赖关系等)。这些文件可以长期保存模块的配置和原型。使用这些文件可重新激活模块，也可在 NPSNET-V 的应用间传递模块的信息。原型文件可代表实体、世界、应用、应用扩展等，有了它可以除去一个模块的多个拷贝；配置文件只是用于已加载的模块。

对于原型文件，Module 抽象类定义了 getPrototype 方法的缺省实现，该方法用来获取原型文件。各模块所在的容器提供了 createModule 方法；该方法根据原型文件，加载并初始化模块、或启动模块；在实例化原型文件对应的模块之前，先根据依赖关系将所有的依赖模块初始化。

对于配置文件，Module 抽象类定义了 getConfiguration 和 applyConfiguration 方法的缺省实现。这两个方法分别使用了 PropertySerializationProvider 和 ConfigurationElementInterpretationProvider提供的服务。前者用来生成相应于已知属性的 XML 配置元素，后者用来解释 XML 配置元素。

2.4 引导过程

NPSNET-V 应用首先由微内核发起一个标准的引导过程：

(1)微内核首先在其 main 方法中将其自身实例化为一个模块：

①注册 NPSNET-V 的协议的处理器。NPSNET-V 的协议有两个：Resource 和 Module。它们的处理器分别是 kernel.jar 中的 resource.Handler 和 module.Handler。

②根据用户的选择，设置代理属性(有些用户在防火墙之后的局域网上，需要通过代理访问 Web；设置这一属性可使 NPSNET-V 通过代理访问 Internet)。

③设置微内核自身的名字(微内核自身的名字一般为“/”)。

④设置微内核的类信息(每个模块有一个指向 ModuleClassDescriptor 对象的引用，该对象用于描述模块的类)。

⑤将微内核注册为配置元素解释服务提供者、模块依赖生成服务提供者、模式位置服务提供者。配置元素解释服务负责将 XML 配置元素运用到模块上。模块依赖生成服务根据模块的原型文件中的〈Dependencies〉元素的内容生成模块依赖对象。模式位置服务将 XML 名字空间(例如“Resource:///org/npsnet/v/kernel/Module.class”)映射到 XML 模式的位置(例如“Resource:///org/npsnet/v/kernel/Module.xsd”)。

⑥生成和注册微内核支持的基本配置解释器，这个解释器即 FundamentalConfigurationElementInterpreter。

⑦生成和注册 Kernel 支持的基本依赖生成器，这个生成器即 FundamentalModuleDependencyFactory。

⑧微内核进入初始化状态。

(2)接着，向微内核运用引导配置文件，从 resource.jar 文件中加载 StandardResourceManager 模块(资源管理者模块)，该模块进入初始化状态，微内核进入启动状态。初始化之后，资源管理者检查本地 NPSNET-V 的 archive 目录，生成所有可获得的资源的列表，这个列表包含每个资源的名称、版本、以及资源(jar 文件)的 manifest 文件中与资源相关的元数据。资源管理者随后尝试与 LDAP(Lightweight Directory Access Protocol)资源服务器连接。当模块根据名称请求资源时，资源管理者查找本地资源及 LDAP 资源服务器上的资源，返回资源最新版本所在的位置。

(3) 运用启动配置文件之后，微内核处理命令行参数。命令行参数包括两类，一类是“-configuration”参数，它使微内核

运用-configuration 所带的配置文件所指的配置；一类是"-prototype"参数，它使微内核将-prototype 所带的原型实例化。这些参数扩展了 NPSNET-V 框架。

(4) 处理完命令行参数之后，微内核将控制权释放给它所加载的模块。

如果应用在执行过程中必须加载其它模块，它们通过基类 Module 和 ModuleContainer 所提供的 API 实现加载。例如，任何模块可以通过它的容器的 createModule 方法生成模块并加载其原型，NPSNET-V 框架由此而得到扩展。

3 实体模型

NPSNET-V 的实体模型是基于 Model-View-Controller (MVC)[7]模式的，它在 NPSNET-V 框架中用模块实现虚拟世界及其中的实体。每个实体的抽象状态(model、模型)与它的外观(view)和控制器(controller)分别在不同的模块中实现。外观用于向用户表达模型，控制器用于控制模型。NPSNET-V 平台中的每个实体包括一个模型模块、任意多个松耦合的外观模块、任意多个松耦合的控制器模块以及一个管理模块(scaffold)。根据应用程序框架状态、模型的信息以及管理模块的内部策略等方面的变化，管理模块生成或销毁外观和控制模块，或是生成子管理模块。

3.1 实体的抽象状态——模型模块

模型模块是 NPSNET-V 的实体的核心模块。它的原型不仅包含实体的物理状态(位置、朝向等)，而且还包含外观模块和控制器模块的原型。因而实体的模型模块的原型完全代表了一个虚拟实体。NPSNET-V 平台中包含了一些实体的模型类。这些实体的模型类包括环境模型、地形模型及普通实体模型等。虚拟环境的开发者可以直接使用这些模型类，也可以把它们作为可扩展的基类。正如其它模块一样，实体的模型模块可被加入框架的层次结构中；例如虚拟世界的模型模块可以包含一些实体模型，而这些实体模型又能包含一些其它的实体模型，即模型之间存在容器层次关系(见图 3)。正如场景图结构(也是一种层次结构)决定了高层图形 API 的绘制行为，模型模块间的容器层次关系影响到模型运作的性质和范围。

3.2 外观(view)模块

外观模块通过与图形库的界面，将实体的模型模块表现给用户观看。外观模块的容器层次结构反映了相应的模型模块的层次结构(见图 3)。例如：在模型模块和外观模块的层次结构中，虚拟世界的模型模块与一个核心外观模块相对应(见图 3)，该虚拟世界模型模块所包含的子模型模块(如虚拟世界中的太阳的模型模块)对应于该核心外观模块所包含的子外观模块(如太阳的外观模块)。当使用场景图 API 时，核心外观模块生成场景图的根，其子外观模块生成和控制场景图的子结点。NPSNET-V 平台包含了 Java3D、Java2D、OpenGL 和文本等外观模块。

3.3 控制器(controller)模块

控制器根据用户输入、网络更新信息、仿真物理交互或其它信息，操纵与它们相应的模型模块。与外观模块一样，控制器模块也有与模型模块相应的容器层次关系(见图 3)；核心控制器模块管理着设备的接口，子控制器模块管理着与它们的模型模块相应的数据。NPSNET-V 平台包含鼠标控制器、键盘控制器、物理控制器、DIS、HLA、XFSP、XRTI 网络通信控制器等。网络控制器模块有其独特之处，它必须生成和控制代表远方主实体(master)的本地假象实体(ghost)。远方主实体是指该实体是由网络上远方主机拥有和控制的。当网络控制器核心模块发现新实体时，它们利用 EntityTypeMappingProvider服务得到适于在本地代表该远方实体的实体模型原型文件。于是控制器核心模块通过实例化该原型，将假象实体加入 NPSNET-V 框架。

3.4 管理模块(scaffold 模块)

当新的实体模型模块出现在模型容器中时，该容器对应的管理模块生成子管理模块。子管理模块管理新实体的外观模块和控制器模块。根据与管理模块相应的模型模块的原型文件、模型模块的父结点相应的外观模块和控制模块、以及它们自身的策略，管理模块生成、更新和销毁外观模块和控制器模块。与每个模型相应的外观原型和控制器原型还包含模式过滤器——由一组模式标志组成的布尔表达式。例如，有的控制器原型带有过滤器：debug&(!ghost)，这将使该原型仅能适用于在调试方式下并不会产生假象实体的模型。

4 典型的 NPSNET-V 架构

典型的 NPSNET-V 架构如图 3 所示，箭头代表容器关系、实线方框代表框架中的模块。其中系统服务模块包括：CommonConfigurationElementInterpreter；StandardResourceManager；StandardEntityTypeMappingProvider；StandardModuleUpgradeProvider；StandardTimeManager；StandardPropertySerializationProvider 等。

这些模块的主要功能是解释配制元素、管理资源、实体类型映射、模块升级服务、时间管理、属性串行化服务等；虚拟环境专用服务模块包括 StandardPhysicalGeometryProvider；StandardWorldManager；StandardTranslationProvider 等，这些模块的主要功能是物理几何服务、虚拟世界管理、物体变换服务等。ModelCore 是 BasicWorld 的实例，它是虚拟世界的模型模块。外观模块层次结构包括 J3dViewCore 子层次结构、J2dViewCore 子层次结构等。控制器模块层次结构包括 PhysicsControllerCore 子层次结构、AgentControllerCore 子层次结构、AwtControllerCore 子层次结构、DisControllerCore 子层次结构等。

5 基于 NPSNET-V 的应用的结构

NPSNET-V 提供了一系列测试应用，这些应用以配置文件的形式给出。每个测试是某个模块或某类功能的演示。以 hla_networking.xml 为例，该配置文件是一个简单 HLA 网络功能测试。它的主要作用是加载另外两个子配置 hla_networking_a.xml 和 hla_networking_b.xml。这两个子配置在不同的窗口中作为不同的客户应用运行。在加载一系列服务模块之后，每个客户生成一个虚拟世界，该世界中有三个物体：镜头、光照和茶壶。接着，相应于世界模型生成一个管理模块。在管理模块(容器)中实例化 Java3D 外观模块(J3DViewCore)，并打开一个绘制窗口。为了加入场景的交互，客户程序生成物理控制器(PhysicsControllerCore)、用户界面控制器(AWTControllerCore)、HLA 控制器(HLAControllerCore)。茶壶的管理模块接着加载一个简单的物理控制器，该控制器负责根据茶壶的速度和加速度来移动茶壶；还加载一个 HLA 控制器，将关于茶壶的状态更新传送给 RTI(Run-Time Infrastructure)。两个客户应用都完成配置后，它们相互发现对方控制的茶壶，于是各自生成一个假象实体以代表对方所控制的实体。运行的结果是两个窗口，当移动一个窗口中的茶壶时，另一个窗口中的假象茶壶以同样方式移动。

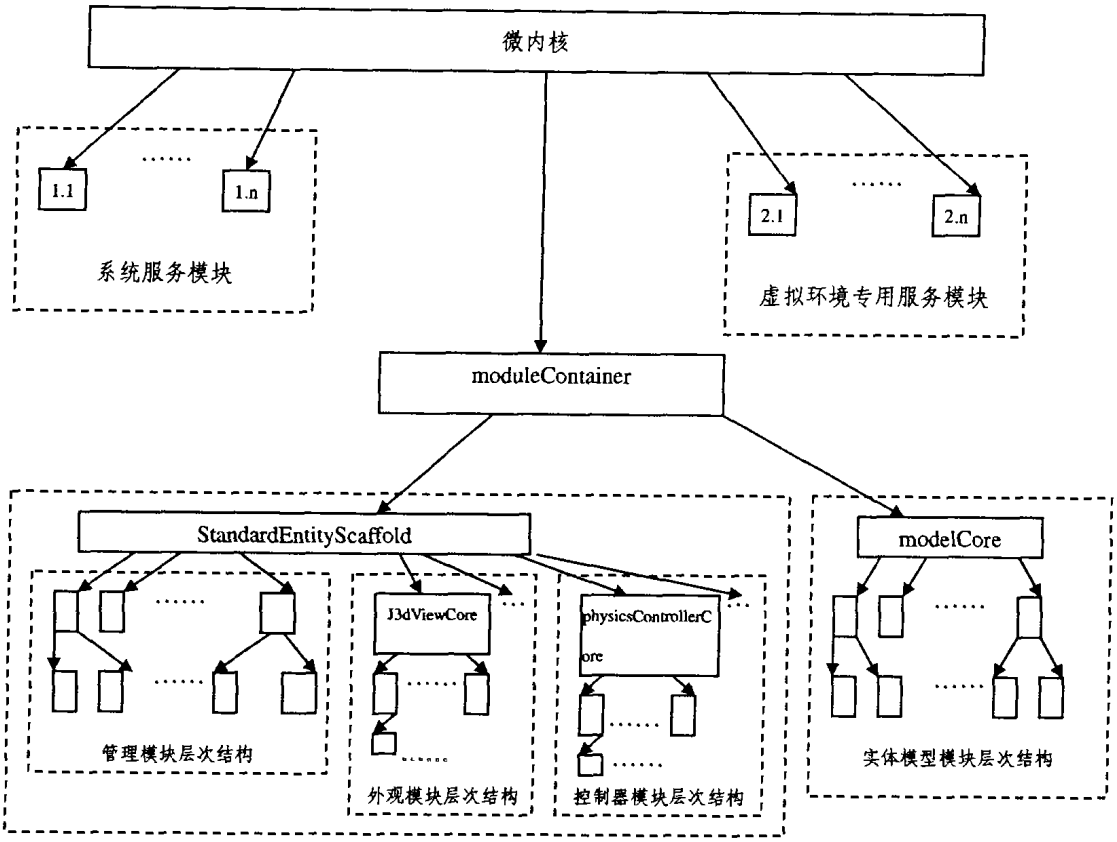

图 3 典型的 NPSNET-V 架构

6 设计启示

通过对 NPSNET-V 的实践和研究，我们得到了以下可供设计大型分布式虚拟环境系统的启示和理念：

(1)不同的应用会有不同的要求。设计大型虚拟环境系统很难归纳出绝对的原则和理念，这是由于不同的虚拟环境应用对虚拟环境系统有不同的要求。例如游戏系统与军事系统的要求各不相同。

(2)采用模块化。“模块内部高的内聚性、模块间的松耦合”(“High Cohesion，Low Coupling”)是设计大型虚拟环境系统应遵守的原则。将可分离的功能分别放在不同的模块中，将模块间的通信降到最低。这样，可以改变应用的一部分而不必担心这一改变对其它部分的影响。

(3)力求可动态扩展[1]。对已经布署了的应用今后应当仍能增加新功能，而且应使系统在运行时就可以加入新代码而不用停止应用的运行。这样系统可以随需求的变化而得到及时进化。

(4)力求可缩放。系统可以对付规模的增长。对于网络虚拟环境，规模的增长是指虚拟环境用户数量的增多、虚拟环境大小的增大、应用的复杂度的增加。

(5)务必可移植。使系统可支持多种平台(硬件和操作系统)。这一点是可缩放性的发展，它将使系统扩展成能够对付具有不同资源(处理能力、内存、网络带宽)的环境。

(6)具有继续性。必须要有措施使得即使虚拟环境中没有参与者，仿真世界仍能继续随时间而发生变化。继续性有两种：主动继续性和被动继续性。对主动继续性，即使没有用户注册进入，计算机也能继续使仿真得以运行。对于被动继续性，则只有当有用户注册进入虚拟环境，仿真才会继续进行。

(7)具有可发展性。虚拟环境系统的设计者应当设计出这样一种系统，使它能够使用任何的图形标准、任何的网络系统(如 HLA、DIS)和任何的图形用户界面而不是只能使用某一种。当改变系统使用的标准时不必重新设计整个系统。这一点尤赖于可移植性的存在。

致谢 衷心感谢美国海军研究生院以 Andrzej Kapolka 为首的 NPSNET-V 项目梯队接纳第一作者为该梯队的研究成员，并允许第一作者在中国刊物上发表有关 NPSNET-V 的部分成果。

参 考 文 献

1 Capps M, McGregor D, Brutzman D, Zyda M. NPSNET-V: A New Beginning for Dynamically Extensible Virtual Environments. IEEE Computer Graphics and Applications, September/October, 2000, 20(5): 12～15

2 Pettifer S, Cook J, Marsh J, West A. Deva3: Architecture for a large-scale distributed virtual reality system. In: Proceedings of the ACM Collaborative Virtual Environments Conference, 2000

3 Kapolka A, McGregor D, Capps M. A Unified Component Framework for Dynamically Extensible Virtual Environments. CVE'02, Bonn, Germany, September 30-October 2, 2002

4 West A J, Hubbold R J. System challenges for collaborative virtual environments. In: D. Snowdon and E. Churchill, eds. Proc. of Collaborative Virtual Environments. Springer-Verlag, 2000

5 Oliveira M, Crowcroft J, Slater M. Component Framework Infrastructure for Virtual Environments. In: Proc. of the ACM Collaborative Virtual Environments Conference, 2000

6 Tanenbaum A S, van Steen M. Distributed Systems—Principles and Practice. Prentice-Hall Inc, 2002

7 Gamma E, Helm R, Johnson R, Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. 1st ed. Addison-Wesley, 1995