高技术通讯 2013 年 第23 卷 第6 期:623 ~630

doi:10.3772/j.issn.1002-0470.2013.06.012

分布仿真系统 HLA 联邦快速开发包的设计与实现①

范林军②* 凌云翔* 王 涛** 韩 韬*

(*国防科学技术大学信息系统工程重点实验室 长沙 410073)

(**德国不来梅大学数学与计算机科学学院 德国 28359)

摘 要 为提高高层体系结构(HLA)联邦开发的快捷性、易用性、可重用性及节约成本,对其运行支撑环境(RTI)接口进行了封装,结合数据公布订购自定义配置文件和 MAP 容器,设计并实现了封装的仿真接口包(SPI)。首先,给出了 SPI 包的框架设计、功能分解结构、时序逻辑及函数接口定义,然后设计了仿真应用数据类的具体规范、联邦可执行文件 XML 和邦元公布订购数据的初始化配置文件,阐述了基于 SPI 的联邦快速开发步骤,最后用一个简单实例验证 SPI。实验表明:相比现有的联邦编程开发软件,SPI 简单高效,用户可自行设计并轻松上手,降低了开发成本,满足了特定的仿真需求。

关键词 分布交互仿真,运行支撑环境(RTI),接口封装,自定义设计,联邦快速开发,高层体系结构(HLA)

0 引 言

高层体系结构(high level architecture, HLA)是用来支持软件模型互操作与组件可重用的分布仿真开发标准[1]。联邦(federation)是 HLA 中的重要概念之一,是指一组交互的联邦成员的集合。HLA 由规则、对象模型模板和接口规范[2]三部分组成,运行支撑环境(run time infrastructure, RTI)是对 HLA 接口规范的具体实现。HLA 已在武器系统仿真[3]、工业控制[4]、网络仿真[5]等众多领域得到广泛应用,而计算机网络技术的发展使得大规模分布仿真成为一种主流趋势[6],这对 HLA 联邦快速开发提出了更高要求。HLA 接口复杂、联邦对象模型模板的编写与维护繁琐等不足导致仿真应用开发效率不高[7],特别是当前建模与仿真的网络化、智能化、普适化等新特点[8],使得传统的 HLA 联邦开发与维护过程有些力不从心。文献[9]完善了 HLA 联邦开发和执行过程(federation development and execution process, FEDEP),文献[10,11]用基本对象模型(base object model, BOM)改进了联邦对象模型模板(object model template, OMT),简化编写,并生成了通用的联邦可执行文件替代品——可扩展标记语言(extensible mark language, XML)。文献[12]提出了一种基于本体的 FOM 框架加快 FEDEP,文献[13]为提高联邦的可重用性与资源的可组合性,设计了联邦社区网络共享机制。上述研究主要集中于 FEDEP 的自身完善,较少论及具体编程开发中的准则与规范。仿真应用开发规范与标准化流程直接影响仿真系统架构效率与性能好坏[14],在 FEDEP 具体实现的C++编程过程中,国内虽有一些机构如神州普惠、海军航空兵工程学院等,通过封装 HLA 中常用函数及制订面向仿真服务的组件结构建模规范,给出了仿真应用全生命周期内的解决方案,加快了联邦开发效率[15-17],但一方面用户需权衡购买此类软件的费用;另一方面,用户的仿真未必使用其全部功能,从而造成资源浪费。本研究只针对分布式仿真平台的搭建,设计了一种简单、易用、可自行设计的仿真接口包(simulation package interface, SPI)。该包在封装 RTI 的基础上,采用 MAP 映射表存储对象类属性和交互参数表,搭配本地配置文件对公布订购数据进行初始化,用 XML 文件替换联邦可执行文件 FED,并制定了仿真数据类编写规范和 SPI 框架下联邦快速开发编程步骤。SPI 包简单高效,用

① 国家自然科学基金(61272336,61005055),国家科技支撑计划(2009BAG12A05)和国家教育部博士点专项基金(20094307120015)资助项目。

② 男,1985 年生,博士生;研究方向:分布交互仿真,面向服务仿真,时空一致等;联系人,E-mail: ljfan_nudt@163.com

(收稿日期:2012-08-20)

户无需购买,可根据设计思路自行封装并轻松上手,降低了开发成本,满足了特定的仿真需求。

1 SPI 设计

1.1 SPI 框架设计

SPI 是对 HLA/RTI 的封装,用一个_SPI 类实现,该类封装了创建联邦执行 CreateFedEx()、加入联邦 JoinFedEx()、撤销联邦 ResignAndDestroy()、公布订购对象类并注册对象实例 PublishSubscribeAndRegisterObject()和公布订购交互类 PublishAndSubscribeInteraction()等 RTI 接口函数[18],同时_SPI 类定义了用于公布的自身对象类_Class、订购对象类集_clasSub、交互类集_clasInt、默认对象类迭代器_citer、属性表迭代器_aiter、RTI 大使、共享数据类_SharedData、初始化公布订购函数 InitData()、RTI 配置函数 ConfigRTI()、RTIambassador 指针生成函数 MakeRTI()、主循环函数 MainLoop()、自定义回调过程函数和封装统一调用函数 RUN()等。_SPI类和 SPIFederateAmbassador 大使类搭配使用,后者为前者的邦元大使类。_SPI类包含文件如表 1 所示。

表 1 _SPI 类文件目录

_SPI 头文件	_SPI 资源文件
_SPI.h	_SPI.cpp
_ClassDef.h	_ClassDef.cpp
_SPIAmb1516.h	_SPIAmb1516.cpp
DataClass.h	DataClass.cpp(非核心)

1.2 功能分解结构

SPI 包功能分解为 8 个组件,如图 1 所示。RTI 初始化组件提供 RTI 的初始化服务,包括联邦文件、联邦名字和联邦类型的初始化等。邦元准备组件(PreFedration)负责读取本地 CLAS.cfg 文件,初始化公布订购数据以及生成有效的 RTI 大使指针等。公布订购注册组件(PSAndSign)提供 RTI 的开启,包括创建联邦、加入联邦、公布订购和注册对象类以及公布订购交互类等。主循环组件(MainLoop)负责属性句柄的构建(用于自身属性更新),进而引起相应订购者属性反射调用;同时构建参数句柄(用于发送交互参数),进而引起相应订购者交互接收调用。RTI 大使组件为邦元提供一个服务调用接口,用于反射自身属性与发送交互参数。邦元回调组件(CallBack)负责回调处理,用于获取订购的属性与交互的参数。退出组件(PreQuit)提供联邦的注销、销毁及邦元的退出。外部接口组件(Interface)用于完成仿真应用数据和控制接口对 HLA 架构下数据和控制机制的转换。

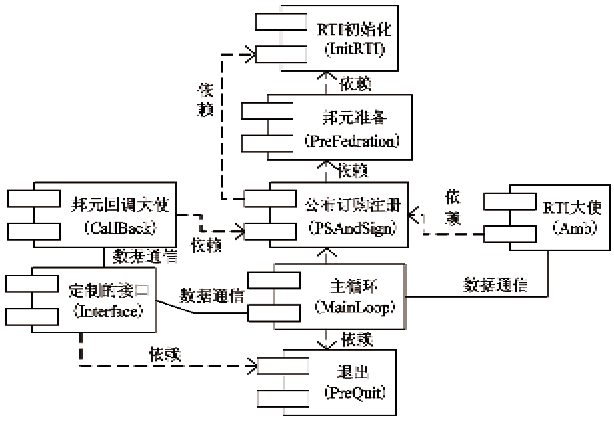

图 1 SPI 包功能组件及内部关联关系

1.3 状态转换及时序

SPI 包内部的执行逻辑及时序如下:在初始化 RTI 后,RTI 大使和仿真邦元分别就绪,然后进入各自的循环逻辑,Mainloop 循环隶属于 MainLoop 线程,负责公布属性与交互参数,RTI 大使循环隶属于 RTI 线程,负责仿真邦元的回调,即获取订购的属性与交互参数。在代理程序开启后,先连接 RTI,之后开启 MainLoop 主线程,RTI 大使线程在连接 RTI 后就自动开启。MainLoop 主循环控制整个程序的执行过程。具体时序逻辑如图 2 所示。

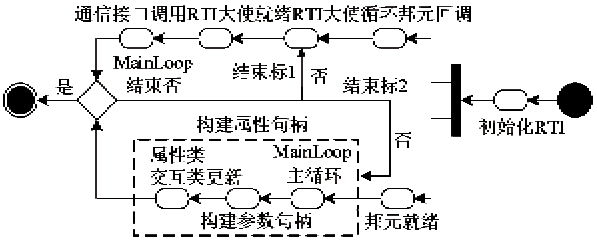

图 2 SPI 接口包状态转换及时序图

1.4 相关定义

(1) 数据类型定义

对象类_Class 为各对象类提供统一封装,按照名称和属性的方式组织数据,是后续操作的基本元素。共享数据类_SharedData 为邦元和邦元大使(RTI-RTIAmb)所共享,用于部分数据的传递。类型属性表_ATTR 为属性和参数提供统一封装,按照名

称和对应数据的方式组织。

(2) 映射表定义

初始化公布订购数据时,Map 映射容器的使用,使得_SPI 内部函数实现过程中对对象类属性与交互类参数的查找变得便利,无需反复繁琐地查找对象模型模板中的数据。映射分为基于名称的映射和基于句柄的映射,名称映射主要实现基于名称的查询、存储和引用;基于句柄的映射主要用于邦元大使回调函数中的查询和引用。Map 映射表定义如下:

```
typedef  map<wstring, _Class>  _CLAS; //名称(N)类型映射
typedef  map<wstring, ObjectClassHandle>  _OCHT;//N 对象类句柄映射
typedef  map<wstring, InteractionClassHandle> _ICHT;  //N 交互类句柄映射
typedef  map<wstring, ObjectInstanceHandle> _OIHT;//N 对象实例句柄映射
typedef  map<wstring, AttributeHandleSet> _AHST;//N 属性句柄集映射
typedef  map<wstring, ParameterHandleSet> _PHST; //N 参数句柄集映射
typedef  map<wstring, AttributeHandle>  _ATHT;// N 属性句柄映射
typedef  map<wstring, ParameterHandle>  _PAHT;// N 参数句柄映射
///参数句柄值二次映射
typedef  map<wstring, ParameterHandleValueMap>_PHVM;
typedef  map<wstring, _ATHT>  _S_ATHT;// N 属性二次映射
typedef  map<wstring, _PAHT>  _S_PAHT;// N 参数二次映射
//对象类句柄名称映射
typedef  map<ObjectClassHandle, wstring>  _ObjectClassMap;
//对象实例句柄名称映射
typedef  map<ObjectInstanceHandle, wstring>  _ObjectInstanceMap;
//交互类句柄名称映射
typedef  map<InteractionClassHandle, wstring>  _InteractionClassMap;
//对象实例对象类映射
typedef map<ObjectInstanceHandle,ObjectClassHandle>  _InstanceClassMap;
```

(3) 配置文件读取函数

ReadData()函数从公布订购配置文件(默认.\CLAS.cfg)读取信息,按照 CLAS.cfg 文件结构存储数据。通过读取目标文件,对类型表进行赋值。数据类型可根据需要自行添加。表 2 为该函数的参数说明。

表 2　文件读取函数 ReadData 参数

参数名	功能说明
classFile	输入文件流
label	当前标签
tag_label	目标标签
clas	写入类型表
_className	当前对象类名
isOnce	类型重复读取标志

(4) RTI 接口封装

CreateFedEx()为创建联邦执行函数,JoinFedEx()函数将 fedAmb 代理邦元加入联邦执行,ResignAndDestroy()函数负责撤销联邦执行,PublishSubscribeAndRegisterObject()函数公布订购对象类,并注册对象实例,PublishAndSubscribeInteraction()负责公布订购交互类。封装函数对应于 RTI 初始化过程,声明为 public,以便外部调用,但在_SPI 类中为内部使用。

(5) 内部函数

如表 3 所示,内部函数主要用于_SPI 类实例直接调用来完成RTI的相关配置(联邦名称、联邦类型和 xml 配置文件[2])、初始化(创建和加入联邦执行、公布和订购对象类和属性、注册对象类实例和公布和订购交互类[19])、主循环(用于属性更新和发送交互及其他操作)及清场。同时为方便测试,引入统一封装函数 RUN()。MainLoop()函数是联邦的核心循环,用于自身对象类属性更新和发送交互类参数,VC++下一套典型的 MFC 程序调用方式如图 3 所示。

表 3　_SPI 内部函数

内部函数定义	函数功能说明
void InitData()	初始化公布订购数据
void ConfigRTI()	配置 RTI
void MakeRTI()	生成有效 RTIambassador 指针
void StartRTI(fedAmb)	初始化并启动 RTI
void MainLoop()	主循环
void Clear()	清场
void RUN(fedAmb)	封装方式统一调用
BE_ATTRPROC()	属性反射 RTI 回调服务
BE_INTPROC()	接收交互 RTI 回调服务

```
char * classFile = "CLAS.cfg";          //公布订购本地配置文件
SPIFederateAmbassador  fedAmb(spi_data);
UINT  ThreadProc(LPVOID lpParam)
{
    spi.RUN(fedAmb);                    //联邦执行封装函数
    return 0;
}
spi.ConfigRTI(federationName,federationFile,federateType);  //RTI配置
spi.InitData(classFile);            //初始化数据
spi._rtiAmb = _SPI::MakeRTI();      //生成有效的RTI指针
AfxBeginThread(ThreadProc,(LPVOID)this);     //线程开启
```

图 3　典型的调用方式(VC++ MFC)

(6) _SPI 大使类

SPIFederateAmbassador 类继承自 NullFederateAmbassador 类[18],封装了 IEEE1516 标准下的

HLA/RTI 标准函数，这里不再详述。需注意：引入外部全局变量，方便调用，如下所示：

```
extern _SPI spi;    //_SPI 类全局声明，此项必需
extern DataClass be; //仿真数据类全局声明，此项自定义
```

发现对象实例中，必须完成对象实例句柄的保存，通过共享数据类_SharedData 的对象实例_data 完成数据传递。对应回调函数的调用如下所示：

```
void SPIFederateAmbassador::reflectAttributeValues ()
{
//调用属性过程回调处理函数
spi.BE_ATTRPROC(be, theObject, theAttributeValues);
}
```

在此函数回调中，应当尽可能调用_SPI 或其它自定义类型相应定义的过程函数，对反射的订购对象类的对象实例更新后的属性进行相应处理，一般用于赋值。这里示例给出的是通过自定义数据类 DataClass 来完成相应数据的赋值和传递。编写程序时尽量不要在回调函数中直接编写代码。

2 SPI 数据配置

2.1 仿真数据类

DataClass 类为扩展类，并非_SPI 框架的核心，但较好的 DataClass 数据类编写规范，可方便公布订购对象属性更新数据和交互参数数据的传递和引用。同时，数据类与_SPI 核心代码中的自定义过程回调函数（见表 3 最后两个）紧密相关。DataClass 类的结构定义如图 4 所示。结构体 HLAData 是公布订购数据的邦元程序中的接入体，结构体 LocalData 用于存储程序执行的控制标志及其他一些相关变量。

```
class DataClass
{
  public:
    struct _HLADATA
    {
      struct _IN inData; //订阅数据存储
      struct _OUT outData; //公布数据存储
    }HLAData; //公布订购HLA数据
    struct _LOCALDATA
    {
      //此处根据需要添加内部共享变量
    }LocalData;
  public:
    DataClass();
    ~DataClass(){};
};
extern DataClass data  //全局类声明
```

图 4 仿真数据类 DataClass 定义

2.2 联邦执行数据文件

大规模分布仿真的发展，要求使用更通用的 XML 传输格式，以满足对象模型的共享、重用和其他应用[15]。XML 已成为仿真互操作领域标准化的数据传输格式，其替代联邦执行数据文件 FED，提高了分布仿真的互操作性、可重用性等[20]。SPIExample.xml 的核心编写如下：

```
<objectClass name="object1" sharing="PublishSubscribe">
    <attribute name="attribute1" dataType="type1" updateType="NA"
  updateCondition="NA" ownership="NoTransfer" sharing="PublishSubscribe" dimensions="NA" transportation="HLAbestEffort" order="TimeStamp"/>
  </objectClass>  //对象类属性
  <interactionClass name="attribute2" sharing="PublishSubscribe"
  dimensions="NA" transportation="HLAreliable" order="Receive">
    <parameter name="Label" dataType="type2"/>
  </interactionClass>  //交互类参数
```

2.3 公布订购配置文件

CLAS.cfg 配置文件由三个标签 <Publish>、<Subscribe> 和 <Interaction> 组成，分别对应程序中需公布、订购的对象类（含属性表）和交互类（含参数表）。为简便，交互类不再细分公布和订购，本配置中定义的交互类默认为既公布又订购，可在交互发送和接受过程函数中选择和过滤。<Interaction> 标签内的“标签”是交互标志，可设为“Data”。对象类属性和交互类参数都按图 5 格式编写。

需注意：一个程序只允许公布（Publish）一个对象类，但可订购（Subscribe）多个对象类。一个程序可公布和订购多个交互类。CLAS.cfg 配置文件，使 HLA 数据的公布与订购变得简便，提高了系统的可扩展性与可重用性，方便后续功能的添加与维护。

```
<Publish>
    对象类名1
    {
      <数据类型1>  <属性名1>
                ……
    }
<Subscribe>
    对象类名2
    {
      <数据类型1>  <属性名1>
    }
<Interaction>
    交互名1
    {
      <数据类型1>  <标签>
    }
```

图 5 CLAS.cfg 配置文件编写规范

3 SPI 包应用开发步骤

基于对 SPI 设计和数据配置的实施,给出 SPI 联邦快速开发步骤(图 6):

(1)根据需求定义联邦对象并设计联邦概念模型,编写符合 HLA1516 标准的 SPIExample. xml 配置文件(替代联邦可执行文件 FED)。

(2)参考 SPIExample. xml 配置文件和仿真系统自身需求编写 CLAS. cfg 公布订购数据本地配置文件(如图 5 所示)。

(3)改写读取数据函数 ReadData( ) (_ Class-Def. cpp),负责联邦公布订购数据时,数据类型的匹配和初始化赋值(参考 CLAS. cfg 文件,以下同)。

(4)编写仿真数据类和公布订购数据结构体(DataClass. h/cpp),主要添加或改写联邦公布和订购的数据。具体参考图 4 示例。

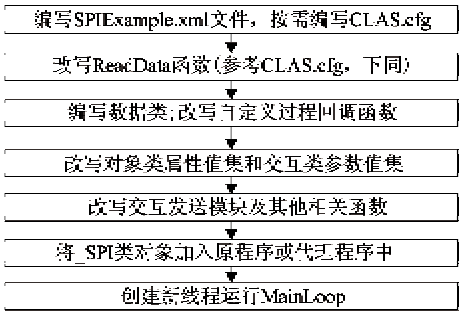

图 6 SPI 下联邦快速开发步骤

(5)按需改写自定义过程函数(表 3 最后两个)。BE _ ATTRPROC( )用于属性反射 RTI 回调服务引用,即订购其他邦元对象类实例属性值更新的数据。BE _ INTPROC( )用于接收交互 RTI 回调服务引用,查询交互类列表,是否为本过程关联的交互类,并订购其他邦元发送过来的交互类参数。

(6)改写属性句柄值集 AHVPS 和参数句柄值集 PHVPS (_ SPI. h/cpp),属性句柄值集用于自身属性的更新,进而引起相应订购者属性反射调用,参数句柄值集 PHVPS 用于改写或添加当前邦元公布的交互类参数数据。

(7)改写交互发送函数 sendInteraction ( )部分(_ SPI. h/cpp),在交互发送函数体所在模块根据需要添加其他交互类参数。改写其他函数(按需改写)。

(8)将_ SPI 类对象加入VC++程序中,创建新线程运行 MainLoop。若仿真基于VC++,则直接嵌入源程序中;若基于其他平台,则需自编VC++代理邦元程序,嵌入该类,运行 MainLoop。

(9)生成可执行文件,开启 RTI 运行邦元。确保 SPIExample. xml、CLAS. cfg 和 HLA. dtd 在该文件目录下。

需注意:初始化函数需要在新建线程外完成。基于 SPI 包的联邦编译执行逻辑如图 7 所示。

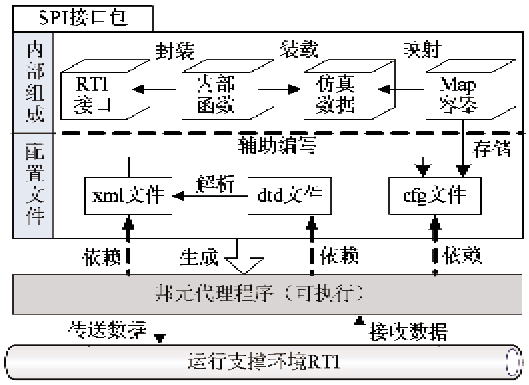

图 7 基于 SPI 包的联邦编译与执行逻辑

4 应用案例

模拟导弹攻击舰艇,导弹采用陆基固定发射,舰艇在某海域沿直线运动,速度可变。设导弹需两次击毁目标,舰艇的初始生命为 100,一次攻击对其毁伤值为 50,二次攻击后舰艇的生命值和速度为 0。设计导弹和舰艇两个邦元,导弹的属性有水平速度 M _ xspeed、垂直速度 M _ yspeed;舰艇的属性有 X 坐标 m _ x、Y 坐标 m _ y、速度 m _ speed、生命值 m _ life。导弹邦元订购舰艇邦元的属性 m _ x、m _ y、m _ speed和 m _ life,所有邦元的交互参数设为 Signal,负责传送发射信号和属性修改指令。在 XP/VC + +2008/Mak-RTI 环境下,基于 SPI 框架设计并实现了各邦元,其开发流程如下:

(1) 编写 xml 文件,关键区段如下:

```
< objectClass name = " VoyageShip" sharing = " PublishSubscribe" >
    < attribute name = " m _ x" dataType = " double" updateType = " NA"
updateCondition = " NA" ownership = " NoTransfer" sharing = " Publish-
Subscribe"
dimensions = " NA" transportation = " HLAbestEffort" order = " TimeS-
tamp" / >
//此处添加其他公布的属性,如 m _ y, m _ speed, m _ life
< /objectClass >
< objectClass name = " CruiseMissile" sharing = " PublishSubscribe" >
    < attribute name = " M _ xspeed" dataType = " double" updateType
```

```
= "NA"
updateCondition = "NA" ownership = "NoTransfer" sharing = "Publish-
Subscribe"
dimensions = "NA" transportation = "HLAbestEffort" order = "TimeS-
tamp" / >
//此处添加其他公布的属性,如 M _ yspeed
< /objectClass >
//交互类参数 Signal
< interactionClass name = "CBSignal" sharing = "PublishSubscribe"
dimensions = "NA" transportation = "HLAreliable" order = "Receive" >
< parameter name = "Signal" dataType = "double" / >
< /interactionClass >
```

(2) 编写 CLAS. cfg 文件,如图 8 所示。

导弹邦元CLAS.cfg编写

```
<Publish >
CruiseMissile
{
double          M_xspeed
double          M_yspeed
}
< Subscribe >
VoyageShip
{
double     m_x
double     m_y
double     m_speed
double     m_life
}
< Interaction >
CBSignal
{
double          Signal
}
```

舰艇邦元CLAS.cfg编写

```
<Publish>
VoyageShip
{
double     m_x
double     m_y
double     m_speed
double     m_life
}
<Subscribe>
CruiseMissile
{
double     M_xspeed
double     M_yspeed
}
<Interaction >
CBSignal
{
double     Signal
}
```

图 8 CLAS 配置文件编写

(3) 改写数据读取函数,匹配数据类型,如下:

```
string x1 = " < Publish > " ; string x2 = " < Subscribe > " ; string x3 =
" < Interaction > " ;
string flag1 = " { " ; string flag2 = " } " ; string c1 ;
c1 = "double" ; / * 在此添加其他支持数据类型 * /
```

(4) 编写仿真数据类和公布订购数据结构体,具体如图 9 所示。

导弹邦元仿真数据

```
struct_IN
{
        double m_x ;
        double m_y;
        double m_speed;
        double m_life;
        double Signal;
};
struct_OUT
{
        double M_xspeed;
        double M_yspeed;
};
```

舰艇邦元仿真数据

```
struct_IN
{
        double M_xspeed;
        double M_yspeed;
        double Signal;
};
struct_OUT
{
        double m_x ;
        double m_y;
        double m_speed;
        double m_life;
};
```

图 9 仿真数据的公布订购结构体

(5) 在 BE _ ATTRPROC ( ) 和 BE _ INTPROC ( ) 函数中分别添加各自邦元订购的对象类属性和交互类参数,具体数据参考 CLAS. cfg。

(6) 同样,在属性句柄值集 AHVPS 和参数句柄值集 PHVPS 中添加各自邦元公布的属性和参数。以导弹邦元的 m _ x 为例,如下:

```
//构建属性句柄值集 AHVPS
for ( _ ATHT: : iterator iter = _ thisAttrNameHandleMap. begin( ) ;
      iter ! = _ thisAttrNameHandleMap. end( ) ; iter + + )
      {
      std: : wstring temp = iter- > first;
      if ( temp = = L" m _ x" )  //对应自身属性
      {
    _ thisAttrValues[ iter- > second ]. setData( ( void * ) &CBdata. m
_ x, sizeof( double ) ) ;
      }
}
//构建参数句柄值集 PHVPS
      for ( _ S _ PAHT: : iterator _ siter = _ interParamNameHan-
dleMap. begin( ) ;
      _ siter ! = _ interParamNameHandleMap. end( ) ; _ siter + + )
      {
      if ( _ siter- > first = = L" CBSignal" )
      {
        for ( _ PAHT: : iterator iter2 = _ siter- > second. begin( ) ;
        iter2 ! = _ siter- > second. end( ) ; iter2 + + )
        {
          if ( iter2- > first = = L" Signal" )
          {
          CBdata. Signal = 50; //自定义
          _ paramValues[ iter2- > second ]. setData( ( void * ) &
CBdata. Signal, sizeof( double ) ) ;
          }
        }
    _ interParamValues[ _ siter- > first ] = _ paramValues; _ param-
Values. clear( ) ;
  }
}
```

(7) 如下步骤按照第 3 节中步骤(7)(8)(9)进行。

图 10给出了系统实现与操作的软件界面。测试表明,SPI 为联邦的快速高效开发提供了新方法。

图 10 导弹攻击舰艇-基于 SPI 的分布仿真演示

5 结 论

本文设计并实现了一种对 RTI 接口进行封装的 SPI 包,给出了其功能结构、内部逻辑、数据配置方案及基于该包的联邦快速开发步骤等。用户可根据 SPI 的框架设计及功能结构、状态转换时序图、开发步骤、联邦编译与执行逻辑和数据配置,在 RTI 支撑下采用VC + + 编程实现。实例表明,相比其他软件,SPI 简单高效,降低了开发成本和周期。SPI 虽提高了联邦开发效率,但仍有不足,如邦元属性添加维护时改写步骤过于繁琐等。下一步,将设计一种面向服务的运行支撑环境,通过资源、服务及节点等配置文件和动态链接库封装各种服务,提供尽可能简单的编程接口和开发步骤,帮助用户以尽可能低的技术门槛开发新的分布式系统或改造原有系统。

参考文献

[1] IEEE 1516-2000. IEEE Standard for Modeling and Simulation (M&S) High Level Architecture [HLA]— FrameWork and Rules, USA, 2000

[2] 周彦,戴剑伟. HLA 仿真程序设计. 北京:电子工业出版社,2002

[3] Fu Y F, Kang F J, Qi J H, et al. Research of dynamic scheduling method for the air-to-ground warfare simulation system based on grid. *Simulation Modeling Practice and Theory*, 2010, 18(8): 1116- 1129

[4] Eusgeld I, Nan C, Dietz S. "System-of-systems" approach for inter-dependent critical infrastructures. *Reliability Engineering & System Safety*, 2011, 96(6): 679-686

[5] Zacharewicz G, Deschamps J C, Francois J. Distributed simulation platform to design advanced RFID based freight transportation systems. *Computers in Industy*, 2011, 62(6): 597-612

[6] Grande E R, Boukerche A. Dynamic balancing of communication and computation load for HLA-based simulations on large-scale distributed systems. *Journal of Parallel and Distributed Computing*, 2011, 71(1): 40-52

[7] Strassburger S, Schulze T, Fujimoto R M. Future trends in distributed simulation and distributed virtual environments: results of a peer study. In: Proceedings of the 2008 Winter Simulation Conference, Miami, USA, 2008. 777-785

[8] 李伯虎,柴旭东,朱文海等. 现代建模与仿真技术发展中的几个焦点. 系统仿真学报,2004,16(9):1871-1978

[9] Lutz R. FEDEP V1.4: An update to the HLA process model. In: Proceedings of the 1999 Winter Simulation Conference, Phoenix, USA, 1999. 1044-1049

[10] Rathnam T, Christiaan J, Paredis J. Developing federation object models using ontologies. In: Proceedings of the 36th Conference on Winter Simulation, Washington, USA, 2004. 1054-1062

[11] Gustavson P, Chase T. Using XML and BOMS to rapidly compose simulations and simulation environments. In: Proceedings of the 36th Conference on Winter Simulation, Washington, USA, 2004. 1467-1475

[12] Suna H B, Fan W H, Shen W M, et al. Ontology-based interoperation model of collaborative product development. *Journal of Network and Computer Applications*, 2012, 35(1): 132-144

[13] Chen D, Turner S J, Cai W T, et al. Synchronization in federation community networks. *Journal of Parallel and Distributed Computing*, 2010, 70(2): 144-159

[14] Codur K B, Dogru A H. Evolution of software development standards in the military domain and effects on software applications. In: Proceedings of the Joint International and Annual ERCIM Workshops on Principles of Software Evolution, New York, USA, 2009. 42-45

[15] 神州普惠. 神州普惠 DWK3.0 版全面支持面向仿真服务组件结构建模规范. http://www.appsoft.com.cn/2012/0219/117.html: 神州普惠公司, 2012

[16] 徐圣良,李仁松,张培珍等. 海军航空兵作战指挥训练系统的关键技术研究. 船舶科学技术,2009,31(5):151-155

[17] 潘长鹏,顾文锦,陈洁等. 分布式航空兵突击作战仿真系统开发. 海军航空工程学院学报,2005,20(5):563-566

[18] IEEE 1516.1-2000. IEEE Standard for Modeling and Simulation (M&S) High Level Architecture [HLA]— Federate Interface Specification, USA , 2001

[19] IEEE 1516.3-2003. IEEE Recommended Practice for High Level Architecture [HLA] Federation Development and Execution Process (FEDEP), USA,2003

[20] Zywicki D J, Prochnow D L, Bhatti A S, et al. Expanding the reach of distributed interactive simulation (DIS) to service-based modeling and simulation environments. In: Proceedings of the Fall Simulation Interoperability Workshop 2011, Orlando, USA, 2011.219-227

Design and implementation of the package for fast development of HLA federation in distributed simulation systems

Fan Linjun*, Ling Yunxiang*, Wang Tao**, Han Tao*
(*Science and Technology on Information Systems Engineering Laboratory,
National University of Defense Technology, Changsha 410073)
(**Faculty of Mathematics and Computer Science, University of Bremen, Germany 28359)

Abstract

To improve the speediness, easy usage and reusability of the federation development of high level architecture (HLA) and reduce the cost of the HLA federation development, some interfaces of HLA's run time infrastructure (RTI) were encapsulated and some simulation package interfaces (SPI) were designed and implemented by adopting the MAP container and user-defined files on publish and subscription relations. Firstly, the SPI package was defined, including its framework, functional structure, time sequence logic and function interfaces. Then, the detail specification of data class for simulation applications, the executive file-XML of federation and initial configuration files for federation data's operations on publishing and subscribing were designed, and the fast development scheme based on SPI was expatiated. Lastly, a simple example was used to validate the SPI. The experimental results show that in comparison with existing softwares of federation development, SPI is simple and effective, and the customers can design it by themselves and use it easily, thus the developing cost can be reduced while it meets the specific simulation requirements.

Key words: distributed interactive simulation, run time infrastructure (RTI), interface encapsulation, user-defined design, federation fast development, high level architecture (HLA)