УДК 681.324

М.А. Волк, А.С. Горенков, Р.Н. Гридель

Харьковский национальный университет радиоэлектроники, Харьков

АРХИТЕКТУРА ИМИТАЦИОННОЙ МОДЕЛИ GRID-СИСТЕМЫ, ОСНОВАННАЯ НА ПОДКЛЮЧАЕМЫХ МОДУЛЯХ

В работе предложена оригинальная архитектура распределенной имитационной среды моделирования GRID-систем, основанная на динамически подключаемых модулях. Обсуждается ее преимущества, недостатки, а также возможная область применения. Рассмотрены особенности реализации описанной архитектуры на основе открытого проекта GRASS (GRID Advanced Simulation System), который разрабатывается в Харьковском национальном университете радиоэлектроники.

Ключевые слова: *модульная архитектура, подключаемые модули, моделирование GRID-системы.*

Введение

В условиях формирования Национальной GRID-инфраструктуры, которая становится частью европейского проекта EGEE [1 – 3], большое значение имеет эффективность объединения значительных вычислительных ресурсов. Одним из наиболее используемых способов повышения эффективности процесса проектирования больших систем является имитационное моделирование. Однако, с учетом размерности решаемых в данном случае задач, организация имитационного моделирования является сложной задачей. Ряд подходов к организации имитационного моделирования в GRID приведено в [4].

Существует большое количество пакетов моделирования GRID-систем. Наиболее распространенными из них являются проекты Bricks [5], OptorSim [6] и GridSim [7]. Детальный сравнительный анализ этих пакетов приведен в [8]. Выводы, приведенные в последней работе, говорят об ограниченности данных систем моделирования. Основные их недостатки – проблемная ориентированность на решение частных задач и специальная реализация алгоритмов на языках высокого уровня (конфигурационные файлы, Java).

В Харьковском национальном университете радиоэлектроники ведется разработка системы имитационного моделирования GRID-систем GRASS (GRID Advanced Simulation System), которая позволит устранить приведенные выше недостатки. Назначение, структура и варианты использования этой системы в научных исследованиях подробно исследованы в [9].

В данной статье приводится описание архитектуры системы GRASS, основанной на плагинах (plug-in) — независимо компилируемых программных модулей, динамически подключаемых к основной программе и предназначенных для расширения и/или использования еѐ возможностей. Данная архитектура положена в основу структуры управляющего модуля моделирования и определяет правила подключения частных имитационных моделей.

Архитектура системы

Система моделирования GRASS имеет модульную архитектуру. Она состоит из ядра и динамически подключаемых модулей (плагинов). Каждый модуль выполняет свою узкоспециализированную задачу, обращаясь при необходимости к другим модулям системы. Ядро предоставляет средства межмодульного взаимодействия, а также обеспечивает начальную загрузку и конфигурацию системы.

Каждый модуль имеет уникальный строковый идентификатор (имя, ID). Он также предоставляет набор интерфейсов взаимодействия с ним для других компонентов системы. Каждый интерфейс модуля также имеет имя и может быть реализован любым количеством модулей. Таким образом, для получения какого-либо интерфейса модуля необходимо знать его имя и имя интерфейса взаимодействия.

Модуль в системе GRASS представляет собой динамически подключаемую библиотеку (dynamic linked library (dll) – в ОС семейства Microsoft Windows, shared objects (so) – в UNIX-подобных ОС), которая реализует фабричный метод (factory method), создающий экземпляр класса модуля. Класс модуля реализует интерфейс Framework::IPlugin, что позволяет универсально работать с ним, не учитывая особенности реализации.

Этот интерфейс включает в себя базовые операции, которые обязан предоставить системе каждый модуль:

- получение имени модуля;
- получение интерфейса с заданным именем;
- инициализация (необязательно);
- завершение работы (необязательно).

Каждый интерфейс, предоставляемый модулем, также должен быть унаследован от стандартного класса Framework::IPluginInterface, который позво-

ляет унифицировать все интерфейсы модулей в системе. Основным запросом к нему является получение имени в виде строкового идентификатора. Общая структурная схема отношений и взаимодействия модулей, их интерфейсов и ядра показана на рис. 1. На диаграмме также представлены классы примера реализации основных интерфейсов подключаемого модуля: Example::Plugin — главный класс реализации плагина, Example::IExample — один из его интерфейсов.

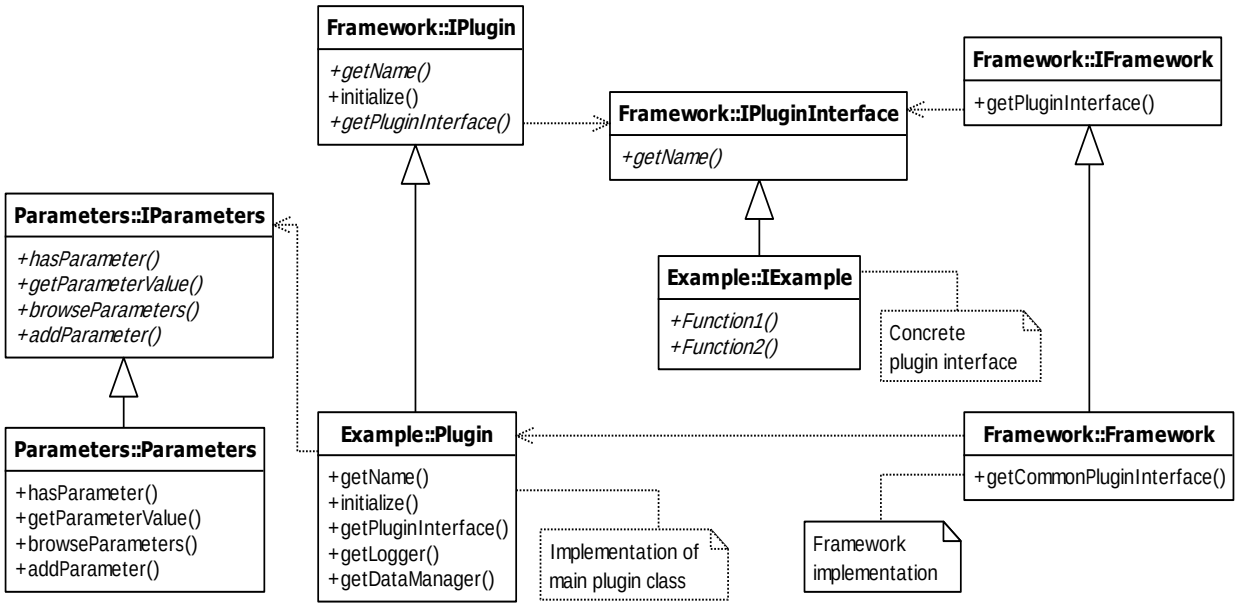

Рис. 1. Структурная схема взаимодействия модулей системы и ядра

Запрос интерфейса одного модуля другим выглядит следующим образом:

1. Запрашивающий модуль вызывает метод ядра getPluginInterface(), передавая ему в качестве аргументов имя модуля и имя его интерфейса.

2. Ядро проверяет наличие модуля с заданным именем и его состояние. Если модуль не найден или уже была неудачная попытка загрузить его, возвращается ошибка. Если это первое обращение к модулю, и он еще не был загружен, выполняется его загрузка в память, инициализация и конфигурирование.

3. Если загрузка прошла успешно, ядро вызывает метод getPluginInterface() и получает требуемый интерфейс модуля по имени, если он поддерживается.

4. При успешном запросе ядро выполняет проверку соответствия имени полученного интерфейса и возвращает его, выполняя преобразование к требуемому типу C++.

Получение доступа к интерфейсам других плагинов происходит через главный интерфейс ядра Framework::IFramework. Он реализуется классом Framework::Framework, который осуществляет обработку конфигурационного файла и начальную загрузку плагинов.

Интерфейс Parameters::IParameters, а также его реализация Parameters::Parameters предназначены для передачи подключаемым модулям начальных параметров, заданных в конфигурационном файле.

Преимущества модульной архитектуры

Гибкость настройки. Система может быть запущена с различным количественным и качественным составом модулей. Например, одно и то же приложение без перекомпиляции может быть запущено в графическом или консольном режиме в зависимости от установленных модулей визуализации. Аналогичным образом можно запускать локальную или сетевую версию приложения, интерактивную версию или версию, которая запускается из командной строки и генерирует отчет. Разумеется, набор модулей должен быть достаточным для выполнения поставленной задачи, т. е. если какой-либо критически необходимый модуль отсутствует, корректная работа системы в данной конфигурации будет невозможна.

Простота модификации. Изменение технологий, разработка новых методов и алгоритмов, а также исправление ошибок в уже готовых реализациях неизбежно приводят к модификации существующих систем. С использованием модульной архитектуры, для обновления системы, достаточно лишь заменить реализацию старого компонента на новую или исправленную, поддерживающую прежние интерфейсы. При этом нет необходимости вносить изменения в остальные компоненты системы.

Упрощение разработки отдельных компонентов. Используя модульную архитектуру, можно легко

реализовать один из основных принципов программирования — декомпозицию задачи на подзадачи, каждая из которых реализуется отдельным модулем. Соответственно, различные модули могут быть реализованы разными разработчиками независимо друг от друга после составления спецификации интерфейсов межмодульного взаимодействия. Кроме того, в большой системе важной также является возможность перекомпиляции одного модуля без сборки других, что значительно экономит время и силы разработчиков.

Упрощение тестирования. Модульная архитектура позволяет легко производить unit-тестирование как отдельно взятого модуля или группы, так и всей системы в целом. Для этого следует реализовать соответствующие тестовые модули, содержащие исходные данные, результаты тестирования, а также механизмы сравнения полученных данных с эталонными, и сконфигурировать систему таким образом, чтобы она использовала их. Кроме того, могут быть разработаны служебные тестовые модули, содержащие общий часто используемый код. Более того, можно также сравнивать между собой результаты работы различных реализаций одного и того же логического элемента (алгоритма) для проверки корректности работы обоих.

Возможность иметь несколько реализаций одного и того же модуля. Модульная архитектура идеально подходит для сравнения различных реализаций одного и того же логического элемента. Это может быть алгоритм защиты, распределения общей нагрузки, расчета какой-либо величины или реализация функциональности для конкретной платформы. Имея несколько модулей, реализующих одну и ту же логику, мы можем поочередно запустить их на одних и тех же данных за счет конфигурации состава системы и, сравнив результаты работы, сделать выводы об их эффективности. Кроме того, модульная архитектура легко позволяет динамически во время исполнения выбирать наиболее эффективный модуль в зависимости от исходных данных или предоставленных ресурсов.

Возможность повторного использования кода. Модульная архитектура позволяет использовать одни и те же наработки в различных проектах, реализованных на основе одинаковых или совместимых ядер. Это могут быть часто используемые компоненты, такие как системы ведения журналов, логов, статистики, менеджеры памяти, компоненты графического интерфейса пользователя (GUI), модули интернационализации и т.д.

Ленивая загрузка — одно из существенных преимуществ модульной системы, которое позволяет экономить оперативную память компьютера, а также время запуска приложения. Это означает, что модуль не будет загружен до тех пор, пока не возникнет необходимость в функциональности, предоставляемой им. Хотя большинство модулей системы следуют этому правилу, возникает необходимость загрузки некоторых модулей при старте системы, чтобы начать выполнять какую-либо работу, вызывая в дальнейшем другие компоненты.

Для создания гибкой модульной системы необходима возможность простого изменения набора плагинов и их параметров без модификации ядра или самих библиотек модулей. Для этого в GRASS используется система конфигурационных файлов на основе XML. Основной конфигурационный файл подключаемых модулей plugins.xml имеет вид, показанный в листинге на рис. 2.

Он описывает взаимосвязи между именами модулей в системе и названиями файлов их библиотек. Кроме того, он позволяет задать параметры, передаваемые модулям при загрузке, которые могут быть использованы для задания режима работы или инициализации внутренних значений.

Некоторые из этих параметров могут также задавать путь к другим файлам, содержащим более детальную информацию о настройках для конкретного модуля.

Выводы

Предложена оригинальная архитектура распределенной имитационной системы моделирования GRID-систем, основанная на подключаемых модулях. Архитектура обладает рядом преимуществ: гибкость настройки, простота модификации, упрощение разработки, тестирования, поддержки и обновления системы.

Перспективы развития системы состоят в создании модулей, реализующих модели элементов GRID-систем, описанных в [9].

Полученные результаты могут быть полезными разработчикам программного обеспечения GRID-систем, специалистам в области разработки распределенного программного обеспечения и создания имитационных моделей.

Список литературы

1. Zgurovsky M.Z. National Ukrainian GRID Infrastructure (UGRID) for Sciences and Educations" / M.Z. Zgurovsky, A.I. Petrenko // Proc. of CSIT-2006, Lvov, 28-30 September 2006. – P. 1-6.

2. Петренко А.И. Национальная Grid – инфраструктура для обеспечения научных исследований и образования / А.И. Петренко // Системные исследования и информационные технологии. – К., 2008. – № 1. – С. 79-92.

3. Integrating ukraine into european grid infrastructure / E. Martynov, A. Petrenko, A. Zagorodny, M. Zgurovsky, G. Zinovjev // Системные исследования и информационные технологии. – К., 2009. – № 2. – С. 3-16.

4. *Волк М.А. Структурная организация поведенческого имитационного моделирования в grid / М.А. Волк // Системи обробки інформації: зб. наук. пр. – Х.: ХУ ПС, 2007. – Вип. 9 (67). – С. 41-45.*

```
<?xml version="1.0" encoding="utf-8"?>
<framework>
        <config>
                <parameter pluginsDirectory = "./plugins/"/>
        </config>
        <plugins>
                <plugin name="AlgorithmLoader" filename="algorithm_loader" loadatstartup="yes" >
                        <parameter TasksCount="10"/>
                        <parameter Algorithm="RandomDistributionAlgorithm"/>
                </plugin>
                <plugin name = "DataManager" filename = "data_manager" />
                <plugin name = "Example" filename = "example" loadatstartup = "no" >
                        <parameter StringToPrint="Hello, world!"/>
                        <parameter Times="3"/>
                </plugin>
                <plugin name = "Logger" filename = "logger">
                        <parameter LogDirectory = "./log/"/>
                        <parameter MessageFormat
                            ="[$(DateTime)][$(MessageLevel)]: $(MessageText)"/>
                        <parameter GlobalMessageLevel = "debug"/>
                        <parameter SectionsMessageLevel = ""/>
                        <parameter StdOutEcho = "all"/>
                </plugin>
                <plugin name = "Queue" filename = "queue" />
                <plugin name = "RandomDistributionAlgorithm" filename = "rda"/>
                <plugin name = "ResourcesController" filename = "rc" />
                <plugin name = "SimpleResourcesManager" filename = "srm" loadatstartup = "yes" >
                        <parameter InputFile="./config/resources_data.xml"/>
                        <parameter ModelConfigFile="./config/resources_data_model.xml"/>
                </plugin>
                <plugin name = "SimpleTasksGenerator" filename = "stm" loadatstartup = "yes">
                        <parameter ModelFile="./config/model.xml"/>
                </plugin>
        </plugins>
    </framework>
```

Рис. 2. Листинг основного конфигурационного файла системы plugins.xml

5. Bricks: A Performance Evaluation System for Grid Computing Scheduling Algorithms [Electronic resource]. – Режим доступа к статье: http://ninf.apgrid.org.

6. Simulating data access optimization algorithms // OptorSim [Electronic resource]. – Режим доступа к статье: http://edg-wp2.web.cern.ch.

7. GridSim: A Grid Simulation Toolkit for Resource Modelling and Application Scheduling for Parallel and Distributed Computing [Electronic resource]. – Режим доступа к статье: http://www.gridbus.org.

8. Коренков В.В. Пакеты моделирования DataGrid [Electronic resource] / В.В. Коренков, А.В. Нечаевский // Системный анализ в науке и образовании. – 2009. – № 1. – Режим доступа к статье: http://sanse.ru.

9. Волк М.А. Структура программного комплекса имитационного моделирования элементов GRID-систем для научных исследований / М.А. Волк // Системи обробки інформації: зб. наук. пр. – Х.: ХУ ПС, 2009. – Вип. 3 (77). – С. 125-128.

Поступила в редколлегию 18.12.2009

Рецензент: д-р техн. наук, проф. каф. О.Ф. Михаль, Харьковский национальный университет радиоэлектроники, Харьков.

АРХІТЕКТУРА ІМІТАЦІЙНОЇ МОДЕЛІ GRID-СИСТЕМИ, ЗАСНОВАНОЇ НА МОДУЛЯХ, ЩО ПІДКЛЮЧАЮТЬСЯ

М.О. Волк, А.С. Горенков, Р.М. Грідель

У роботі запропонована оригінальна архітектура розподіленої імітаційної середи моделювання GRID-систем, заснованої на модулях, що динамічно підключаються. Обговорюються її переваги, недоліки, а також можлива область застосування. Розглянуті особливості реалізації описаної архітектури на основі відкритого проекту GRASS (GRID Advanced Simulation System), що розробляється в Харківському національному університеті радіоелектроніки.

Ключові слова: *модульна архітектура, модулі, що підключаються, моделювання GRID-системи.*

THE ARCHITECTURE OF THE GRID SIMULATION SYSTEM BASED ON PLUG-INS

M.O. Volk, A.S. Gorenkov, R.M. Gridel

The original architecture of the GRID simulation system based on the plug-ins is proposed. Its advantages, disadvantages and application area are discussed. Implementation details of the described architecture are represented based on the open source project GRASS (GRID Advanced Simulation System), that is developed in the Kharkiv national university of radioelectronics.

Keywords: *module architecture, connected modules, design of the GRID-system.*