УДК 681.324

М.А. Волк, А.С. Горенков, О.В. Тучин

Харьковский национальный университет радиоэлектроники, Харьков

ИСПОЛЬЗОВАНИЕ СКРИПТОВ ДЛЯ ПОВЫШЕНИЯ ГИБКОСТИ ИМИТАЦИОННОЙ МОДЕЛИ GRID-СИСТЕМЫ

В работе предложен способ повышения гибкости программной системы имитационного моделирования GRID за счет использования скриптов, расширяющих ее базовые возможности. Обсуждается преимущества и недостатки данного подхода, а также особенности его применения для реализации компонентов системы. Рассмотрены особенности архитектуры на основе открытого проекта GRASS (GRID Advanced Simulation System), который разрабатывается в Харьковском национальном университете радиоэлектроники.

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

Введение

В условиях формирования Национальной GRID-инфраструктуры, которая становится частью европейского проекта EGEE [1, 2, 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 приведен 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. Одним из существенных недостатков этих систем является неоптимальное соотношение производительности к гибкости системы. Большинство из них написаны на Java (Bricks, OptorSim, GridSim), что обеспечивает большую гибкость за счет понижения производительности. Некоторые реализованы на компилируемых языках (C++, Pascal, Fortan), однако такая реализация усложняет их модификацию и ухудшает возможности конфигурации.

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

В данной статье приводится описание архитектуры подсистемы GRASS, которая использует скриптовые средства для реализации некоторых модулей, не влияющих критически на общую производительность системы и подвергающихся наиболее частым изменениям. Обсуждаются преимущества и недостатки данного подхода.

Анализ инструментальных средств программирования

Все языки программирования можно разделить на 2 типа:

- компилируемые (C/C++, Pascal, Fortran и другие);
- интерпретируемые (Python, Ruby, Perl и другие).

Основным преимуществом компилируемых языков программирования является высокая скорость выполнения написанных на них программ. Однако платой за это является более высокая сложность их использования (например, ручное управление памятью, необходимость генерации бинарных файлов для каждой поддерживаемой платформы, недостаточная их совместимость) и меньшая гибкость (например, отсутствуют механизмы рефлексии или интроспекции). Интерпретируемые языки позволяют переносить написанные с их использованием программы на множество платформ (фактически под все, под которые реализованы интерпретаторы этих языков). В них также часто применяется нестрогая типизация, встроенные сложные типы данных, механизмы автоматической сборки мусора и другие удобные программные абстракции, однако производительность программ, написанных на них, часто оказывается неудовлетворительной. Отдельно следует выделить байт-код ориентированные языки (например, Java или языки из семейства технологии .NET), которые занимают среднее положение между компилируемыми и интерпретируемыми. Они пытаются найти компромисс между скоростью выполнения и удобством разработки программных систем. Частично им это удается, однако иногда даже их производительности

недостаточно для удовлетворительной работы системы. Кроме того, они вносят в работу приложения случайные задержки, связанные с работой внутренних механизмов (например, запуск сборщика мусора), которые никак нельзя предсказать на этапе разработки, и которые могут быть нежелательны на некоторых фазах работы приложения.

Как показывает практика, критически важными с точки зрения производительности являются лишь небольшие фрагменты программы, в то время как скорость выполнения остальных не оказывает существенного влияния на общую производительность системы. В этом случае может быть использована следующая архитектура: наиболее сложные (в вычислительном плане) операции или компоненты должны быть реализованы на компилируемом языке программирования, а логика управления ходом вычислений и взаимодействия компонентов может быть реализована в виде отдельных скриптов. Такая концепция уже давно используется при разработке компьютерных игр, когда сложные операции по созданию сцен, физическому моделированию и искусственному интеллекту противника реализованы в виде бинарных модулей, имеющих скриптовый интерфейс, а логика игры (сценарии уровней, общее поведение объектов в игровом мире) задается при помощи скриптов.

Применение скриптовых средств для создания среды моделирования GRID

Применение скриптовых средств для создания среды моделирования GRID-системы совместно с архитектурой, основанной на подключаемых модулях [4,10], позволяет добиться еще большей гибкости. Например, отдельные компоненты системы моделирования GRID (очередь заявок, контроллер ресурсов, менеджер распределения заявок) могут быть реализованы в виде отдельных подключаемых модулей, имеющих открытый скриптовый интерфейс. При запуске системы загрузчик считывает стартовый скрипт, который, в свою очередь, выполняет создание и связывание компонентов в единую систему через их интерфейсы. При этом он может использовать различные реализации одного и того же компонента (например, обычная или приоритетная очередь заявок, менеджер распределения заявок с различными алгоритмами распределения) в зависимости от настроек, заданных пользователем. Кроме того, загрузчик может создавать несколько копий одного и того же модуля (например, для моделирования систем с внутренней и внешней очередями или для моделирования параллельных систем обработки заявок).

Реализацию функциональности в виде отдельных скриптов также удобно использовать для быстрого прототипирования новой функциональности. Например, для быстрого создания и проверки различных пользовательских моделей генерации заявок и ресурсов. При этом, в дальнейшем, если модель будет успешной, она может быть реализована в виде отдельного двоичного модуля для достижения более высокой производительности. Однако, следует отметить, что реализация (и даже прототипирование) на скриптах таких сложных в вычислительном плане компонентов системы, как менеджер распределения заявок, может не принести желаемых результатов из-за критичности фактора производительности их работы. Поэтому реализация подобных модулей с использованием скриптовых средств нецелесообразна.

Скрипты также удобно применять для сбора и накопления статистических данных по работе различных компонентов системы моделирования. При этом собранная статистика может как просто сохранятся в файл или базу данных, так и обрабатываться “на лету” и использоваться для управления работой среды моделирования GRID. Кроме того, эти данные могут быть исследованы при помощи различных алгоритмов статистической обработки или отображаться пользователю в режиме реального времени. Простейшими примерами таких статистических данных для среды моделирования GRID могут быть текущее число заявок в системе, коэффициент использования ресурсов, число поступивших и обработанных заявок за различные периоды времени работы системы.

Еще одна успешная область применения скриптовых средств – преобразование форматов входных и выходных данных. Дело в том, что различные системы моделирования используют свои форматы хранения заявок, ресурсов и результатов распределения. Некоторые используют стандарт JDL (Job Description Language — Язык Описания Заданий), некоторые – свой формат, основанный на XML (eXtensible Markup Language – расширяемый язык разметки), а некоторые – просто текстовые файлы с определенным разделителем. Кроме того, иногда возникает необходимость использования лог-файлов работы реальных GRID-систем в качестве входных данных, чтобы проверить работу конкретной модели в реальных условиях. Они также имеют свой формат, который не совпадает с перечисленными выше. Хотя суть содержания всех этих файлов сходна (все они содержат описания имеющихся заявок или ресурсов и описание их свойств), для использования их между системами необходимы специальные модули преобразования форматов данных. Их удобно реализовывать с помощью скриптов, т.к. в этом случае их можно будет легко адаптировать в случае изменения форматов данных или для приспособления к сходным форматам других подобных систем.

Отдельно следует выделить также применение скриптовых средств для выполнения автоматизированного компонентного тестирования самой GRID-системы в процессе ее разработки. При этом стартовый скрипт заменяется на специальную тестовую версию, которая загружает только тестируемый модуль и окружение, требуемое для его работы. После

этого, скрипт вызывает методы тестируемого модуля и проверяет результаты их выполнения. Такая схема позволяет производить компонентное тестирование одного модуля или даже целой подсистемы из нескольких моделей без внесения изменений в их исходный код и перекомпиляции.

Детали реализации архитектуры с использованием скриптов на примере среды моделирования GRASS

Среда моделирования GRASS имеет модульную структуру, поэтому поддержка скриптовых средств реализована в виде следующих модулей системы:

- модуль поддержки скриптового движка (script engine);
- модуль загрузки скриптов в качестве модулей системы;
- модули трансляции интерфейсов плагинов в скриптовые объекты.

Схема реализации поддержки скриптовых средств изображена на рис. 1.

В системе GRASS в качестве скриптового языка был выбран Qt Script (основан на стандарте ECMA-262, является родственником JavaScript), т.к. для него есть поддержка в библиотеке Qt, которая активно используется в проекте для кросс-платформенной реализации базовых компонентов. Однако, интерфейсы к другим скриптовым языкам (например, Python, Lua) выглядят похожим образом, так что описанные механизмы могут быть применены и для интеграции с ними.

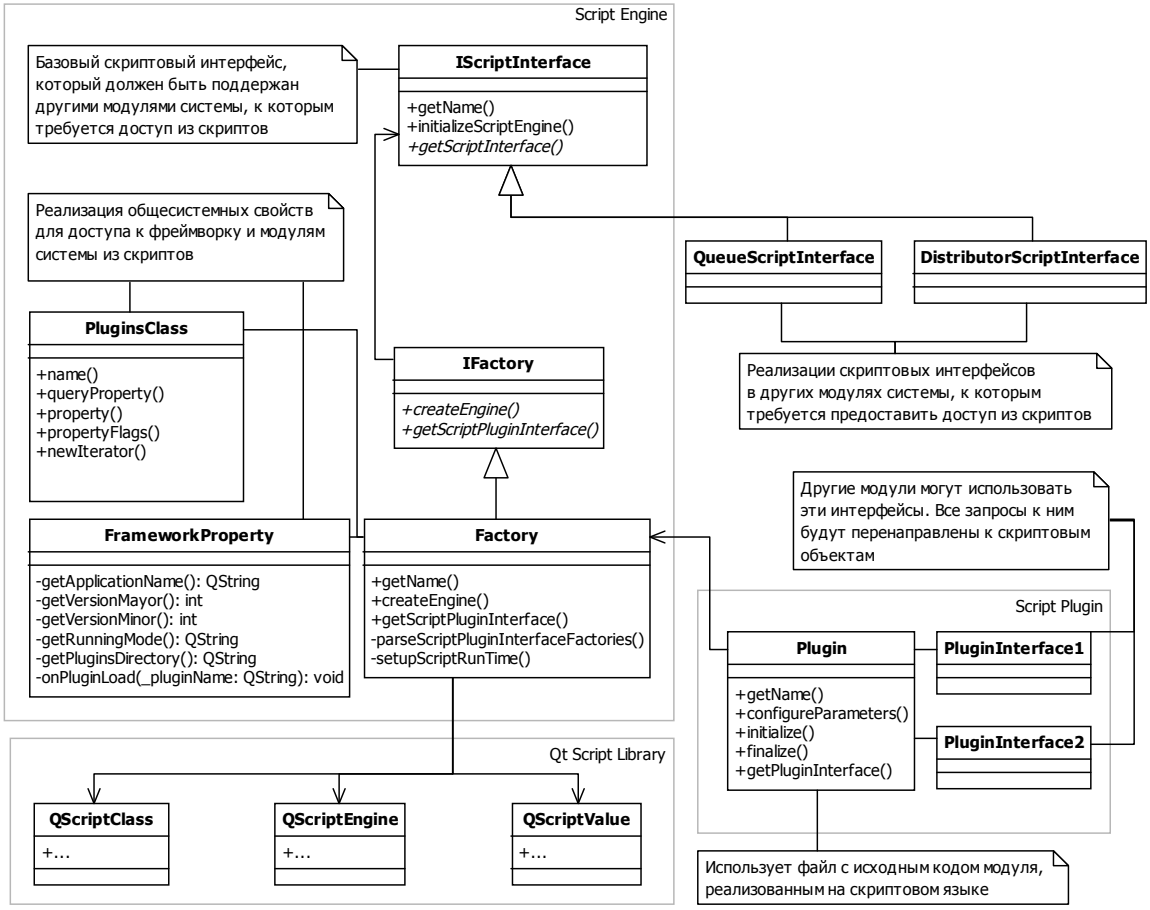

Рис. 1: Схема поддержки скриптов в GRASS

Центральным компонентом GRASS Script Engine является фабрика (Factory) создания объектов скриптовых движков. Она загружается в единственном экземпляре и обеспечивает инициализацию объектов скриптовых движков по требованию других модулей системы. В инициализацию входит создание Qt Script Engine и настройка системных свойств (реализующихся с помощью PluginsClass, FrameworkProperty и других аналогичных классов), которые обеспечивают доступ к Framework и другим модулям из скрипта. При поступлении запроса к какому-либо стороннему модулю (например, Queue), загружается соответствующий плагин (если он не был загружен ранее) и у него запрашивается интерфейс поддержки скриптинга (для модуля Queue на диаграмме это QueueScriptInterface). Если модуль загружен и требуемый интерфейс поддерживается, через него создается скриптовый объект QScriptValue, который обрабатывает поступивший и все последующие запросы к текущему модулю из скрипта.

Модуль GRASS Script Plugin обеспечивает считывание, запуск на выполнение и обработку ошибок при выполнении скриптов, используя для своей работы GRASS Script Engine. Он может быть загружен в нескольких экземплярах, если в виде скриптов реализовано несколько подсистем.

Примеры реализации некоторых модулей на скриптах приведены в листингах 1 и 2.

Листинг 1. Пример реализации подсистемы генерации отчетов в GRASS.

```
fileName = "./last_data/report.html";
num = 1;
startTime = new Date();

// Функция записи в файл информации о возникшем событии
function dumpDistribution(resource, task) {
    currTime = new Date();
    appendFile(fileName, "<TR>\n");
    appendFile(fileName, format("\t<TD>{0}</TD>\n", currTime-startTime));
    appendFile(fileName, format("\t<TD>{0}</TD>\n", num));
    appendFile(fileName, format("\t<TD>{0}</TD>\n", task.id));
    appendFile(fileName, format("\t<TD>{0}</TD>\n", task.memory));
    appendFile(fileName, format("\t<TD>{0}</TD>\n", resource.name));
    appendFile(fileName, format("\t<TD>{0}</TD>\n", resource.memory));
    appendFile(fileName, "</TR>\n");
    num = num + 1;
}
function initialize() {
    // Подписка на обработку интересующих событий
    resourcesController = plugins.ResourcesController;
    resourcesController.onRunTask.connect(dumpDistribution);
    // Запись в файл заголовка отчета
    startFile(fileName, "<html><head><title>Report</title></head><body> ");
    appendFile(fileName, "<h1>GRASS Simulation report!</h1>\n");
    appendFile(fileName, "<table border=\"1\">\n<TR>\n");
    appendFile(fileName, "\t<TH>Simulation time (ms)</TH>\n");
    appendFile(fileName, "\t<TH>No</TH>\n");
    appendFile(fileName, "\t<TH>Task ID</TH>\n");
    appendFile(fileName, "\t<TH>Task Memory Requrement</TH>\n");
    appendFile(fileName, "\t<TH>Resource Name</TH>\n");
    appendFile(fileName, "\t<TH>Resource Memory</TH>\n");
    appendFile(fileName, "</TR>\n");
}
function finalize() {
    // Запись в файл завершающей информации для отчета
    appendFile(fileName, "</table>\n");
    appendFile(fileName, "</body></html>\n");
}
```

Листинг 2. Пример реализации простейшего алгоритма распределения заявок в виде скрипта.

```
// Реализация объекта алгоритма распределения в виде скрипта
distributionAlgorithm = {
    "run" : function()
    {
        for (task in plugins.queue.tasks)
        {
            for (resource in plugins.resourcesController.resources)
            {
                if (plugins.distributor.checkStdParams(resource, task))
                {
                    plugins.distributor.runTask(resource, task);
                }
            }
        }
    },
}

// Регистрация объекта алгоритма распределения в качестве скриптового интерфейса плагина
pluginInterfaces = {
    "IDistributionAlgorithm": distributionAlgorithm,
}
```

В листинге 1 представлен пример скрипта, который реализует генерацию простейшего отчета о результатах моделирования GRID-системы (в формате HTML). Скрипт работает следующим образом: после загрузки плагина вызвается функция `initialize()`, которая регистрирует свои обработчики на возникающие события в системе (событие распределения задания на ресурс) и генерирует стандартный заголовок HTML-документа. При каждом распределении задания на ресурс системы вызывает зарегистрированный ранее обработчик `dumpDistribution()`, который добавляет интересующую пользователя информацию в отчет. Перед выгрузкой плагина вызывается функция `finalize()`, которая добавляет стандартное окончание HTML-документа.

В листинге 2 реализуется простейший алгоритм

распределения, работающий по принципу FIFO. Скрипт реализует предопределенный системой интерфейс `IDistributionAlgorithm`. Метод `run()` этого интерфейса вызывается внутренним компонентом Algorithm Loader, когда требуется распределить задания по ресурсам системы (когда есть новые поступившие задания, когда освобождаются занятые или подключаются новые ресурсы). Реализация этого метода в скрипте ищет нераспределенные задания, а также свободные ресурсы. Если пара свободный ресурс-задание найдена, с помощью системного метода `checkStdParams()` проверяется, удовлетворяются ли ресурсом требования задания по быстродействию процессора, объему памяти и платформе (также можно проверить и другие специфичные для алгоритма параметра ресурса и задания). Если требования удовлетворены, задание распределяется на ресурс с помощью метода `runTask()` компонента Distributor, и продолжается дальнейший поиск свободных ресурсов и нераспределенных заданий.

Выводы

Применение скриптовых средств повышает гибкость разрабатываемой системы, однако может значительно повлиять на ее общую производительность. Как показывает практика, производительность отдельных компонентов системы не так существенна, поэтому они могут быть реализованы с использованием скриптов, чтобы упростить их дальнейшую разработку, поддержку и сопровождение. В то же время, критические компоненты системы должны быть реализованы в бинарном виде и предоставлять скриптовый интерфейс взаимодействия. Применение описанной архитектуры также выгодно для автоматизированного тестирования. Перспективы развития системы состоят в расширении скриптовых интерфейсов к различным компонентам системы моделирования GRID, а также поддержке различных скриптовых языков программирования.

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. – 2008. – №1. – С. 79-92.

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

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

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

6. Simulating data access optimization algorithms // OptorSim. [Электронный ресурс]. – 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. [Электронный ресурс]. – Режим доступа к статье: http://www.gridbus.org.

8. Коренков В.В. Пакеты моделирования DataGrid / В.В. Коренков, А.В. Нечаевский // Системный анализ в науке и образовании. – 2009. – Вып. 1. [Электронный ресурс]. – 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. – Х.: ХУ ПС, 2009. – Вип. 3(77). – С. 125-128

10. Волк М.А. Архитектура имитационной модели GRID-системы, основанная на подключаемых модулях / М.А. Волк, А.С. Горенков // Системи обробки інформації. – Х.: ХУ ПС, 2010. – Вып. 1(82). – С. 17-20.

Поступила в редколлегию 13.05.2011

Рецензент: д-р техн. наук, проф. С.Г. Удовенко, Харьковский национальный университет радиоэлектроники, Харьков.

ВИКОРИСТАННЯ СКРИПТІВ ДЛЯ ПІДВИЩЕННЯ ГНУЧКОСТІ ІМІТАЦІЙНОЇ МОДЕЛІ GRID-СИСТЕМИ

М.О. Волк, А.С. Горенков, О.В.Тучин

У роботі запропоновано метод підвищення гнучкості програмної системи імітаційного моделювання GRID за допомогою використання скриптів, що розширюють її базові можливості. Обговорюються переваги та недоліки цього підходу, а також особливості його застосування для реалізації компонентів системи. Розглянуті особливості архітектури на основі відкритого проекту GRASS (GRID Advanced Simulation System), що розробляється в Харківському національному університеті радіоелектроніки.

Ключові слова: *скрипти, модульна архітектура, моделювання GRID-системи.*

USING SCRIPTS TO IMPROVE FLEXIBILITY OF THE GRID SIMULATION SYSTEM

M.A. Volk, A.S. Gorenkov, O.V. Tuchin

The original way to improve flexibility of the GRID simulation system with scripts that are extending its base features is proposed. Its advantages, disadvantages and particular application for implementation of different components are discussed. Details of the 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: *scripts, module architecture, design of the GRID-systems.*