УДК 681.324

С.С. ОЛЕНДАРЕНКО, М.А. ВОЛК, Р.Н. ГРИДЕЛЬ

АРХИТЕКТУРА ПОДСИСТЕМЫ ВИЗУАЛИЗАЦИИ ДАННЫХ РАСПРЕДЕЛЕННОЙ ИМИТАЦИОННОЙ СИСТЕМЫ МОДЕЛИРОВАНИЯ GRID

Описываются исследования для решения проблемы разработки максимально гибкой и расширяемой архитектуры подсистемы визуализации данных, предназначенной для распределённой имитационной системы моделирования GRID. Разрабатывается оригинальный подход в реализации задачи создания графического интерфейса на основе распределённой модульной архитектуры с применением событийно-управляемой системы отношений между объектами типа «подписчик-слушатель».

1. Постановка задачи

В современных программных системах одним из основных элементов является интерфейс пользователя, который оказывает сильное влияние на скорость освоения приложений, время выполнения прикладных задач, стоимость программных разработок, производительность отдельных сотрудников и целых организаций. В последнее время появилось множество научных публикаций в этой предметной области [1,2]. Наиболее актуальной и обсуждаемой является построение визуального интерфейса распределенных и параллельных систем [3-5], ярким представителем которой является GRID-инфраструктура [6].

Цель данной работы – описание модели визуального интерфейса для распределенной имитационной системы моделирования GRID-инфраструктуры – GRASS (GRID Advanced Simulation System), разрабатываемой в Харьковском национальном университете радиоэлектроники [7,8].

2. Состав системы визуализации

В распределенной имитационной модели GRID функциональность визуализации данных не является непосредственно частью структуры системы, а наряду с другими подсистемами выделена в отдельные модули: “Windows Manager” и множество модулей-адаптеров.

Модуль “Windows Manager” представляет собой основу подсистемы визуализации, которая при запуске системы в графическом режиме создает главное окно, а также предоставляет методы, позволяющие другим компонентам системы регистрировать собственные графические элементы, панели инструментов и пункты меню. “Windows Manager” также занимается размещением и группировкой этих графических элементов внутри главного окна.

Модули-адаптеры, являютщиеся вспомогательными компонентами системы, создают графические элементы, регистрируют их в главном окне с помощью модуля “Windows Manager” и обеспечивают взаимодействие графического интерфейса пользователя с основными модулями системы, а также своевременное обновление данных, предоставляемых этими модулями, в главном окне.

3. Модуль “Windows Manager”

Как и любой модуль в системе, “Windows Manager” поддерживает и реализует интерфейс Framework::IPlugin [8]. Вся логика работы менеджера окон расположена непосредственно в классе WindowsManager, с которым связан класс плагина (plug-in) [8] отношением агрегирования (рис. 1) [9].

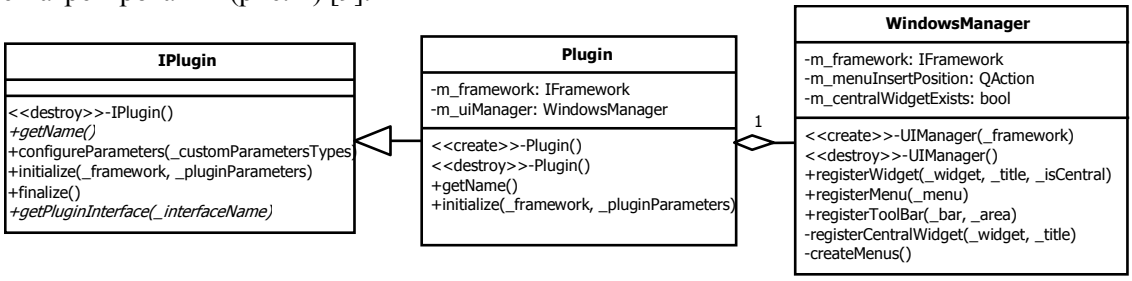

Рис. 1. Диаграмма классов модуля «Windows Manager»

Система конфигурируется таким образом, что при запуске в графическом режиме плагин менеджера окон загружается автоматически, создавая при этом главное окно и ожидая регистрации графических элементов другими плагинами. Класс Plugin — основной класс, описывающий данный модуль в системе распределенного имитационного моделирования GRID. Данный модуль реализует интерфейс IPlugin, описание методов которого представлено ниже. Метод initialize() вызывается при запуске системы и загрузке данного модуля. Все действия, которые необходимо выполнить до непосредственного начала работы модуля, описаны в этом методе.

Для получения имени модуля необходимо вызвать метод getName(). Метод getPluginInterface() в свою очередь позволяет получить один из интерфейсов модуля по его имени: в качестве параметра он принимает название интерфейса. Такая двухуровневая схема доступа к интерфейсу модуля позволяет одному модулю поддерживать несколько интерфейсов для взаимодействия.

Класс Windows Manager наследуется от класса QMain Window библиотеки Qt [10], а также предоставляет интерфейс, описанный в IWindows Manager и содержащий нижеприведенные методы.

Метод registerWidget() позволяет любому другому модулю системы зарегистрировать в главном графическом окне системы собственный элемент. Ответственность за расположение этого графического элемента ложится на Windows Manager, в то время как обновление состояния должен производить модуль-адаптер на своё усмотрение.

Методы register Menu() и register ToolBar() предоставляют возможность другим модулям системы добавлять в главное окно собственные пункты меню и собственные панели инструментов соответственно. Как и в случае с регистрацией графического элемента, обработка событий для добавляемых меню/панелей осуществляется модулем, инициирующим регистрацию.

Таким образом, WindowsManager играет пассивную роль и занимается лишь правильным отображением и размещением графических элементов в главном окне, а инициировать процесс добавления этих элементов должен непосредственно каждый модуль в отдельности.

4. Модули-адаптеры

Модули-адаптеры используются для того, чтобы снизить связанность (coupling) системы и повысить связность (cohesion) каждого отдельно взятого модуля, следуя правилу эффективной разработки «low coupling – high cohesion». В стандартной ситуации каждому модулю системы помимо выполнения своей основной работы пришлось бы «заботиться» и о выводе информации на экран оператора системы. С добавлением нового функционала сложность модуля растет экспоненциально, т.е. уровень связности падает, а уровень связанности, наоборот, возрастает вследствие корреляции этих двух параметров, что приводит к затруднению дальнейшей модернизации и сопровождения системы.

Модули-адаптеры, призванные решить эту проблему, представляют собой дополнительную прослойку между графической и аналитической подсистемами, т.е. играют роль связующего звена.

Для каждого модуля, чьи данные необходимо выводить в графическом режиме работы системы, создается дополнительный модуль-адаптер, который так же, как и все, наследует и реализует интерфейс IPlugin. При этом сам модуль-адаптер не предоставляет собственного интерфейса, так как всю работу выполняет на этапе загрузки. Таким образом, система должна быть сконфигурирована так, чтобы все необходимые модули-адаптеры автоматически загружались при старте. Сама же подсистема регистрации элементов графического интерфейса, т.е. модуль Windows Manager, описанный выше, загружается «по первому требованию», что позволяет избавиться от дополнительных накладных расходов на процессорное время и память, необходимые для процесса загрузки и инициализации, в случае если система сконфигурирована таким образом, что ни один модуль-адаптер не запускается автоматически.

Каждый модуль-адаптер может иметь различное внутреннее устройство, однако должен выполнять как минимум одну функцию из трех возможных: инициировать регистрацию графического элемента, инициировать добавление пунктов меню и/или инициировать добавление панели инструментов.

Все регистрируемые компоненты хранятся внутри самого модуля-адаптера, поэтому типичная реализация модуля, выполняющего все три функции, подразумевает наличие внутреннего класса Widget, а также набора действий, инкапсулированных в классах QAction, на основе которых могут создаваться как пункты меню, так и кнопки панелей инструментов.

При старте системы и последующей инициализации модуля-адаптера вызывается его конструктор, который обеспечивает создание всех необходимых графических компонентов и их регистрацию в менеджере окон, после чего за их отображение отвечает “Windows Manager”. Обновление данных в уже зарегистрированном и отображаемом графическом элементе производит модуль-адаптер. Связь адаптера с модулем, предоставляющим данные, производится на основе паттерна (pattern) проектирования “Наблюдатель” (Observer). Согласно этому паттерну, модуль, предоставляющий данные, реализует метод addListener(), а модуль-адаптер реализует метод notify() и подписывается на события, происходящие в основном модуле. Соответствующая диаграмма классов приведена на рис. 2.

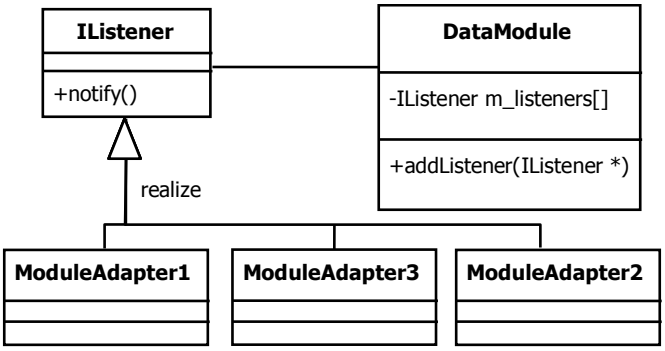

Рис. 2. Диаграмма отношений основного модуля с модулями-адаптерами на основе паттерна проектирования “Наблюдатель”

Опишем схему работы: модуль-адаптер подписывается [11] на события в предоставляющем данные модуле, посредством вызова метода addListener(), где в качестве “наблюдателя” или “слушателя” указывает себя. Этот модуль сохраняет слушателя в соответствующей коллекции, и когда данные обновляются, он вызывает для всех зарегистрированных в коллекции слушателей (модулей-адаптеров) метод notify(). Оповещенный модуль-адаптер повторно считывает данные из основного модуля через открытый интерфейс.

Такой подход дает возможность не только присоединять несколько модулей-адаптеров к одному основному модулю, позволяя выводить информацию от одного и того же модуля в различном виде, но и присоединять один модуль-адаптер к нескольким основным модулям, чтобы выводить общую информацию, поступившую от каждого из них.

Выводы

Предложена новая архитектура визуального интерфейса для распределённой имитационной системы моделирования GRID-инфраструктуры — GRASS. Описанное решение позволило выделить визуализацию данных в отдельную подсистему, уменьшить связанность модулей в системе и увеличить их внутреннюю связность. Одним из основных достоинств архитектуры является возможность подключения новых модулей со своими оригинальными функциями визуализации без перекомпиляции всего проекта. Гибкость разработанной архитектуры позволяет расширять или модифицировать модули визуализации в целях проведения более сложных экспериментов.

Научная новизна результатов состоит в получении модели взаимодействия подсистемы визуализации данных и построении пользовательского графического интерфейса с остальными компонентами распределённой имитационной системы моделирования GRID, реализуемого на основе принципа «подписчик — слушатель».

Практическая значимость результатов заключается в возможности быстрого построения графических интерфейсов пользователя для имитационной распределённой системы моделирования GRID, используя при этом модульный подход, что позволяет наращивать элементы подсистемы визуализации данных по мере необходимости без опасений поте-

рять контроль над сложностью проекта из-за многочисленных внутренних межкомпонентных связей в распределённой имитационной системе моделирования.

Список литературы: 1. *Грибова В.В., Тарасов А.В.*. Модель онтологии предметной области "графический пользовательский интерфейс" // Информатика и системы управления. Амурский государственный университет. Благовещенск, 2005. №1(9). С.80-90. **2.** *Манаков Д.В.* Анализ параллельных визуальных технологий // Вычислительные технологии. 2007. Т. 12, № 1. С. 45-60 **3.** *Brodlie K., Brooke J., Chen M. et al.* Visual Supercomputing Technologies, Applications and Challenges, Eurographics. STAR Reports. 2004. P. 37–68. **4.** *Байдалин А.Ю.* Опыт разработки полнофункционоальной системы визуализации параллельного программирования // Тр. 35-й Регион. школы-конф. "Проблемы теоретической и прикладной математики". Екатеринбург. 26.01–30.01. 2004. С. 303–308. **5.** *Karpov A.N.* Data visualization on parallel computer systems // Тр. 15-й Междунар. конф. по компьютерной графике и зрению. ГрафиКон-2005, 20–24 июля 2005. Новосибирск. 2005. С. 211–214. **6.** *Thiebaux M., Tangmunarunkit H., Czajkowski K., Kesselman C.* Scalable Grid- Based Visualization Framework. Technical Report ISI-TR-2004-592, USC/ Information Sciences Institute, June 2004. **7.** *Волк М.А.* Структура программного комплекса имитационного моделирования элементов GRID-систем для научных исследований // Системи обробки інформації. Вип. 3(77). 2009. С.125-128 **8.** *Волк М.А., Горенков А.С.* Архитектура имитационной модели GRID – системы, основанная на подключаемых модулях // Проблеми інформатики і моделювання. Матеріали дев'ятої міжнародної науково-технічної конференції. Х.:НТУ "ХПІ", 2009. 92 с. **9.** *Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж.* Приемы объектно-ориентированного проектирования. Паттерны проектирования. СПб.: Питер, 2008. 37 с. **10.** *Шлее М.* Qt4. Профессиональное программирование на C++. СПб: БХВ-Петербург, 2007. 553 с. **11.** *Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж.* Приемы объектно-ориентированного проектирования. Паттерны проектирования. СПб.: Питер, 2008. 280 с.

Поступила в редколлегию 11.03.2010

Олендаренко Сергей Сергеевич, студент ХНУРЭ. Научные интересы: разработка архитектур распределенных приложений, построение компиляторов, функциональные языки программирования, ГИС-системы. Хобби: программирование игр, велосипедный спорт. Адрес: Украина, 61166, Харьков, пр. Ленина, 14, тел. +380 (66) 494-11-67.

Волк Максим Александрович, канд. техн. наук, доцент кафедры ЭВМ ХНУРЭ. Научные интересы: распределенное имитационное моделирование, GRID – системы, системное программирование. Хобби: туризм, спортивное ориентирование, автомобили. Адрес: Украина, 61166, Харьков, пр. Ленина, 14, тел. +380 (50) 343-42-90.

Гридель Ростислав Николаевич, аспирант ХНУРЭ. Научные интересы: распределенные имитационные системы моделирования. Хобби: компьютерная техника. Адрес: Украина, 61166, Харьков, пр. Ленина, 14, тел. +380 (57) 7021-354.