УДК 658.012.011.56

МЕТОДЫ РЕАЛИЗАЦИИ ВЗАИМОДЕЙСТВИЯ ПРОГРАММНЫХ МОДУЛЕЙ, РЕАЛИЗОВАННЫХ НА ЯЗЫКЕ JAVA, С ФУНКЦИОНАЛЬНЫМ ПРОГРАММНЫМ ОБЕСПЕЧЕНИЕМ

В.М. Левыкин, М.А. Жигалов.
Харьковский национальный университет радиоэлектроники.

В рамках данной работы решается задача разработки метода взаимодействия программного модуля, реализованного на языке программирования Java, с другими компонентами, составляющими функциональное программное обеспечение, и реализованными на языках программирования, компиляторы которых формируют исполняемые модули под операционную систему Windows.

***Ключевые слова**: модуль, программа, Java, COM, JNI, Windows.*

Введение

При профессиональной разработке программного обеспечения автоматизированных систем (АС) практически каждый раз возникает необходимость построения модульной структуры программной реализации. Подобная мера, как правило, обуславливается характером АС, масштабом АС, а также индивидуальными требованиями к системе.

Возникают ситуации, когда при разработке программного обеспечения недостаточно одного языка программирования для реализации системы. Причинами этого являются преимущества и недостатки различных языков и их реализаций. В этом случае возникает необходимость стыковки программных модулей для их регламентированного взаимодействия.

Эта проблема особенно остро стоит при использовании языка программирования Java. С одной стороны, данный язык обладает рядом значительных достоинств, которые способствовали его широкому распространению, а именно: межплатформенная переносимость, объектно-ориентированный подход, бесплатность [1]. Множество разработчиков, использующий различные платформы, нашли ряд преимуществ в использовании технологии Java. Одна из них – «напиши один раз, используй везде» позволяет писать программные продукты на одной платформе и использовать их на другой. Широкое распространение этого языка повлекло за собой разработку большого количества Java-ориентированных технологий, которые, в свою очередь, значительно расширили область применения Java, а также способствовали включению поддержки этого языка во многие сторонние продукты.

С другой стороны существует ряд недостатков, которые делают этот язык недостаточным при решении масштабных задач, и возникает необходимость использовать другие средства.

Анализ проблемы и постановка задачи

Одним из основных недостатков Java, как интерпретируемого язика, является недостаточное быстродействие. Кроме того, так как Java-программы функционируют в рамках своей виртуальной машины, становятся недоступными некоторые системные механизмы и технологии. На основании этого возникает необходимость в создании программных модулей, реализованных средствами других языков, позволяющих создать исполняемый файл операционной системы, а, следовательно, повысить скорость выполнения критических участков и использовать системные механизмы и функции операционной системы.

Взаимодействие между программным модулем, реализованным в Java, и программными модулями, реализованными средствами других языков, осуществляется с помощью технологии Java Native Interface (JNI) [2]. Однако данную технологию нельзя рассматривать как инструментальное средство, применимое в любом проекте, осуществляющем взаимодействие с программными модулями, реализованными средствами других языков программирования. Использование технологии JNI является достаточно неудобным, так как части модулей,

осуществляющие взаимодействие как со стороны Java, так и со стороны внешних программных модулей, должны быть спроектированы для каждого конкретного случая индивидуально. Описанный недостаток накладывает значительные ограничения на использование данной технологии. Это привело к появлению программных оболочек, предоставляющих программистам удобный интерфейс для пользования технологией JNI, например программный продукт JNIWrapper.

Он выделяется из ряда аналогичных инструментов для решения поставленной задачи большой функциональностью: поддержкой API-функций операционной системы и внешних библиотек, возможностью работы с CALLBACK-функциями, технологией COM, поддержкой реализации всех типов данных, используемых в операционной системе, в том числе структур и указателей.

Рассмотрим механизм действия этого инструмента. Структура работы JNIWrapper и его взаимодействия с внешними программными модулями представлена на рис. 1.

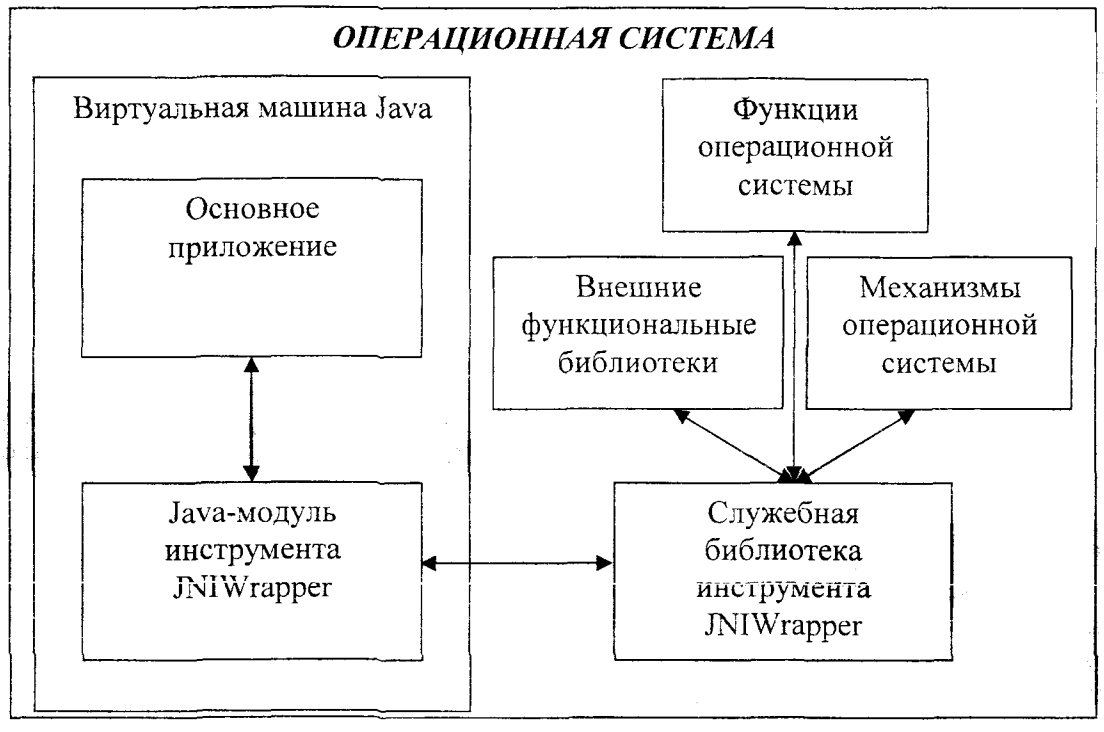

Рис. 1. Общая структура работы JNIWrapper и его взаимодействия с внешними программными модулями.

Программный модуль, реализованный средствами Java, функционирует в рамках виртуальной машины Java (JVM). При необходимости получения доступа к внешним функциям или сервисам осуществляется обращение к ним через интерфейс, предоставляемый Java-модулем инструмента JNIWrapper, который в этом случае является частью основного выполняемого приложения. Реализация данного интерфейса берет на себя взаимодействие с библиотекой JNIWrapper посредством технологии JNI, передачу и получение через нее данных для вызова функций из внешних библиотек, а также данных, контролирующих процесс выполнения внешних функций и необходимых для механизма обработки ошибок. Библиотека JNIWrapper реализует полученные управляющие воздействия применительно к внешним и системным библиотекам или механизмам операционной системы.

Основным преимуществом данного инструмента является предоставление программисту удобного интерфейса для взаимодействия с внешними программными модулями: реализация Java-эквивалентов для типов данных, используемых в операционной системе Windows, унифицированный вызов внешних функций, однотипный механизм обработки

предусмотренных ошибок, а также обеспечение возможности использования внешней библиотеки, написанной не под конкретный проект и без учета правил написания библиотек для использования технологии JNI.

Недостатком в данном случае является загрузка в оперативную память дополнительной внешней библиотеки – библиотеки JNIWrapper. Однако затраты времени для выполнения этой операции незначительны, в то время как полученные преимущества достаточно велики, поэтому потерей быстродействия можно пренебречь.

Но существует еще один, более серьезный недостаток. Согласно механизму использования динамически подключаемых библиотек в операционной системе Windows, библиотека загружается непосредственно в адресное пространство процесса, который ее использует, то есть становится его частью [3]. В случае использования инструмента JNIWrapper в память основного процесса, а именно виртуальной машины Java, выполняющей основную программу, будет загружено две внешние библиотеки. Этот аспект приведен на рис. 2.

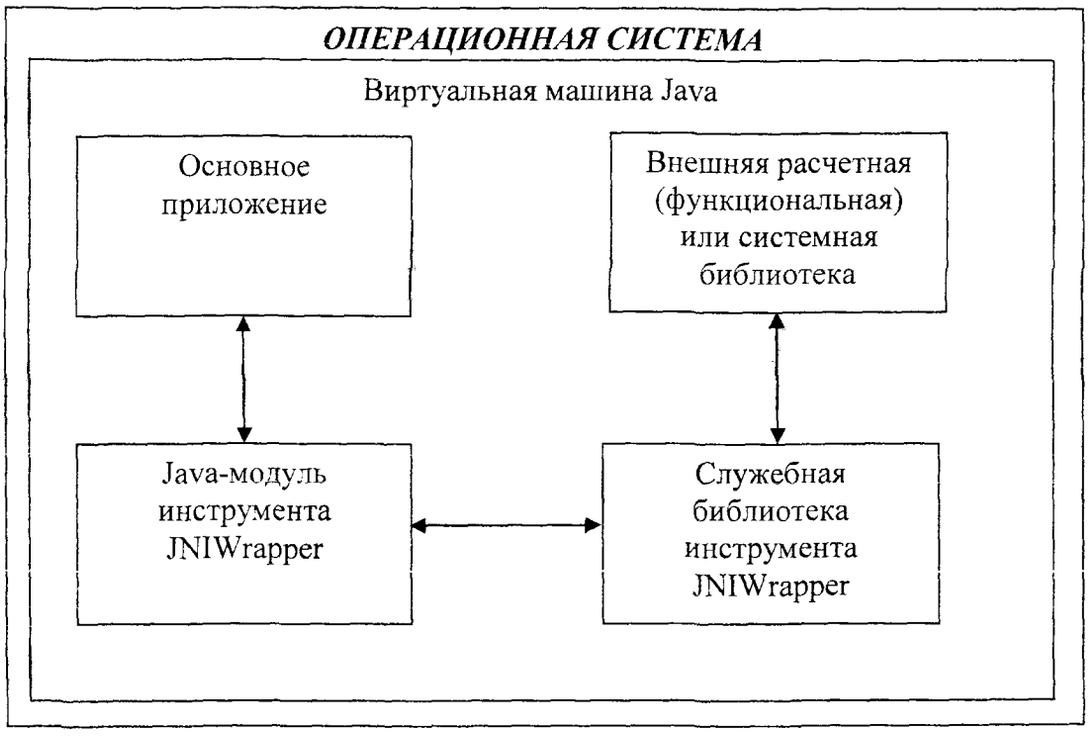

Рис. 2. Структура работы JNIWrapper с разделением на процессы в операционной системе Windows.

Поэтому, при возникновении непредвиденной критической ошибки внутри *функциональной библиотеки, не обрабатываемой непосредственно внутри нее, операционная* система будет рассматривать эту ошибку как ошибку всего процесса и уничтожит его. Также данная ситуация возникает в случае, когда функциональная библиотека по какой-либо причине использует большой объем ресурсов операционной системы, но не освобождает их. Подобные действия операционной системы не дают возможности основной программе прореагировать на ошибку, и сама программа также перестает функционировать.

В реальной ситуации, когда в подавляющем большинстве случаев специалисты по Java не являются специалистами по другим языкам, а, следовательно, разные компоненты функционального программного обеспечения создаются разными группами разработчиков, а иногда используются и функциональные библиотеки сторонних производителей, вероятность ошибки или несогласования каких-либо моментов очень велика.

Решение задачи и основные результаты

Решением данной проблемы является перенос расчетной части рассмотренной архитектуры в отдельный внешний процесс, который будет контролироваться основной программой. Применение подобного решения приведет к снижению быстродействия данной архитектуры, но перенос расчетной библиотеки во внешний процесс – это единственное возможное решение проблемы сбоев основной программы при использовании библиотек сторонних производителей. В то же время временные затраты при дополнительном детальном анализе и корректировке непосредственно самого механизма взаимодействия, как показывает практика, можно свести к минимуму.

Остается нерешенной задача выбора механизма взаимодействия между основной программой и подконтрольным ей приложением, взаимодействующим с расчетной библиотекой. Для организации этой связи кратко рассмотрим реализацию межпроцессного взаимодействия, предоставляемую операционной системой Windows.

В операционной системе представлен ряд механизмов межпроцессного взаимодействия, которые могут быть использованы для решения различных задач. Основными механизмами являются буфер обмена, технологии DDE, FILE MAPPING, MAILSLOT, PIPE (каналы), а также применение сообщения WM_COPYDATA [4].

На основании проведенного исследования механизмов межпроцессного взаимодействия был выбран механизм каналов, который, с одной стороны, расположен на достаточно низком системном уровне и, следовательно, показывает высокое быстродействие, а, с другой стороны, *обеспечивает надежное соединение с синхронной передачей данных, а также обеспечивает* возможность взаимодействия в рамках локальной компьютерной сети.

Результирующая схема организации взаимодействия программных модулей, реализованных на языке Java, с другими компонентами функционального программного обеспечения может быть представлена на рис. 3.

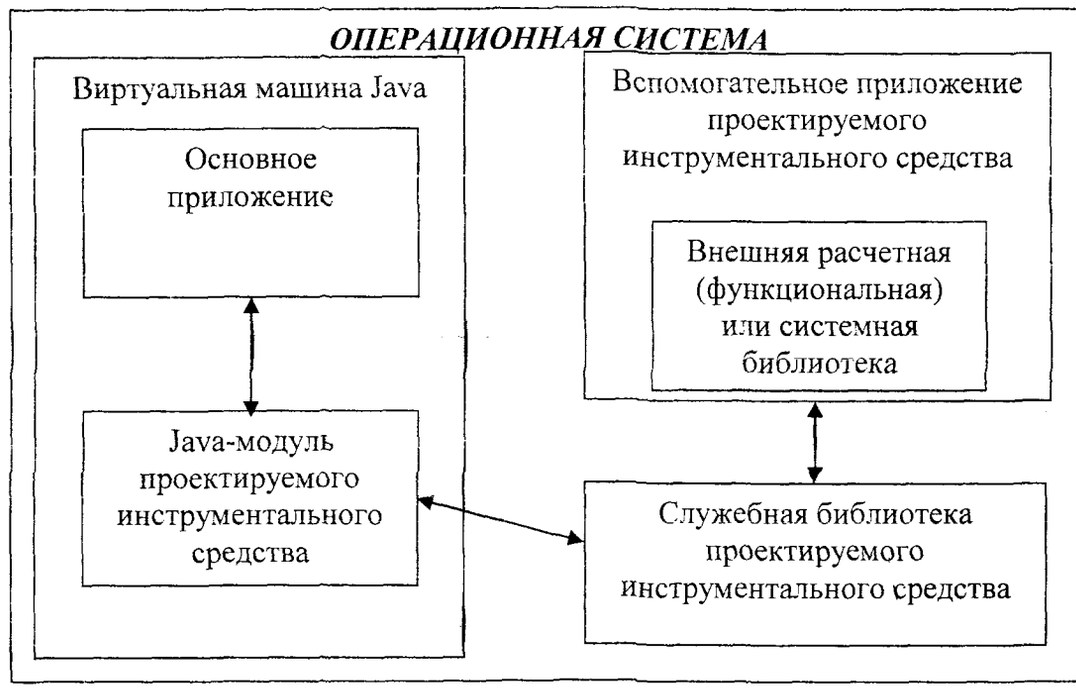

Рис. 3. Результирующая схема организации взаимодействия программных модулей.

На этой схеме показано взаимодействие приложения, реализованного средствами Java, с внешними компонентами посредством интерфейса, предоставляемого Java-частью предложенного инструментального средства. Данный интерфейс предоставляет эквиваленты типов данных, используемых в операционной системе, и стандартизированный механизм вызова функций и обработки ошибок.

Далее запрос к внешней расчетной библиотеке передается от Java-части инструментального средства служебной библиотеке. Служебная библиотека осуществляет контроль наличия в памяти компьютера работающего вспомогательного приложения, при необходимости его запуск, а также передачу данных и управляющих команд, прием данных и контроль состояния внешнего приложения. Взаимодействие между этими двумя компонентами осуществляется при помощи технологии каналов.

В функции вспомогательного приложения входит загрузка расчетной библиотеки, вызов ее функций и возвращение результатов в соответствии с командами инструментальной библиотеки. При каждом взаимодействии инструментальной библиотеки со вспомогательным приложением осуществляется передача уникальных идентификаторов. Применение идентификаторов и поддержка этим приложением механизма многопоточности позволяют использовать ведение параллельных расчетов с использованием данной расчетной библиотеки в рамках всего комплекса функционального программного обеспечения.

Выводы

Таким образом, применение данной схемы позволяет решить проблемы взаимодействия программных модулей, реализованных средствами Java, с другими компонентами функционального программного обеспечения. Реализация этой схемы обеспечивает высокую надежность функционирования всего комплекса и приемлемое быстродействие, дает возможность создать собственное бесплатное инструментальное средство для решения подобных задач, в то время как рассмотренный программный продукт JNIWrapper является коммерческим средством.

ЛИТЕРАТУРА:

1. Блох Дж. Java. Эффективное программирование. – М.: Лори, 2002. – 224 с.
2. Смирнов Н.И. Java 2. – М.: Три Л, 2000. – 320 с.
3. Вильямс А. Системное программирование в Windows 2000 для профессионалов. – СПб.: Питер, 2000. – 624 с.
4. Джонсон М. Харт. Системное программирование в среде Win32. – М.: Вильямс, 2001. – 464 с.

Получено редакцией 10.02.2005.

Левыкин Виктор Макарович, доктор технических наук, профессор, директор Института компьютерных информационных технологий Харьковского национального университета радиоэлектроники, зав. кафедрой информационных управляющих систем.

Жигалов Максим Александрович, студент.

Харьковский национальный университет радиоэлектроники.