УДК 004.42

Л. С. Плинер

АРХИТЕКТУРА РАСПРЕДЕЛЕННОЙ СИСТЕМЫ ПОДДЕРЖКИ СИМВОЛЬНЫХ ВЫЧИСЛЕНИЙ НА БАЗЕ МАТЕМАТИЧЕСКОГО ПАКЕТА GAP

1. Введение

1.1. О пакете GAP

Математический пакет GAP (сокращение от Groups, Algorithms, Programming – группы, алгоритмы, программирование) является мощным инструментом символьных вычислений, ориентированным главным образом на запросы алгебры [1]. В его состав входит интерпретатор собственного языка программирования GAP, который используется одновременно для реализации алгоритмов и взаимодействия с пользователем. Хотя в названии пакета фигурирует слово «группа», возможности языка GAP не ограничивают его применение алгоритмами для групп, и в библиотеке функций пакета в большом количестве имеются алгоритмы для других алгебраических структур, например полугрупп, колец, решеток и др.

Отличительной чертой GAP всегда являлась его нацеленность на поддержку исследовательской деятельности в дискретной алгебре: в ведущих научных журналах регулярно появляются публикации, в которых GAP используется для построения и анализа контрпримеров, «угадывания» формулировок утверждений и гипотез и т. п. Другим немаловажным отличием GAP является его распространение на условиях публичной лицензии GNU: пакет является бесплатным, открытым и расширяемым, никто не должен платить за его использование, а исходный код пакета открыт и доступен для изучения и внесения в него изменений; кроме того, GAP работает под управлением большого числа операционных систем. Благодаря отмеченным выше качествам GAP занял заметное положение в своей области применения: по подсчетам создателей пакета, сегодня насчитывается от 1 тыс. до 2 тыс. активных пользователей GAP, что составляет значительную долю от общего числа алгебраистов-исследователей.

Разработка пакета была начата в Рейнско-Вестфальском техническом университете (Ахен) в 1985 г. под руководством профессора Йоахима Нойбюзера; в проекте GAP участвовали его студенты и аспиранты. В 1988 г. была

выпущена вторая версия пакета, а в 1994 – третья. Постепенно изменился состав участников проекта GAP: коллектив разработчиков стал интернациональным. В 1997 г. координационным центром проекта GAP становится один из старейших в Шотландии Университет Сэнт-Андрюс, а в 1999 г. выходит четвертая версия пакета.

1.2. Структура пакета GAP

Дистрибутив четвертого, последнего на сегодняшний день выпуска четвертой версии пакета состоит из ядра, библиотеки функций, библиотеки данных, документации и произвольного набора пакетов дополнений (рис. 1).

Ядро GAP выполнено на языке Си и представляет собой исполняемый файл. В его состав входят интерпретатор языка GAP и среда разработки. После запуска ядра на базе встроенных в интерпретатор структур данных и функций при помощи библиотеки функций и библиотеки данных, написанных на языке GAP, создается среда выполнения. Среда разработки обеспечивает возможность передавать команды на выполнение интерпретатору, просматривать результат и историю их выполнения, а также обращаться к документации.

Работа ядра происходит по схеме read– evaluate– write, где:

- в состоянии read выполнение происходит в среде разработки и состоит в считывании команды из стандартного потока ввода;
- в состоянии evaluate выполнение происходит в интерпретаторе и состоит в синтаксическом разборе введенного текста и выполнении содержащихся в нем инструкций;
- в состоянии write происходит вывод текста в стандартный поток вывода.

По окончании выполнения команды происходит переход в состояние read.

В ядре GAP имеются функции для запуска исполняемых файлов и обмена данными с ними через псевдотерминал. Это позволяет реализовать критичные по времени алгоритмы вне ядра GAP и обращаться к сервисам, работа с которыми не реализована в ядре. В документации пакета GAP такие исполняемые файлы называются расширениями GAP (GAP extensions).

Для создания программ-надстроек, базирующихся на ядре GAP, предусмотрен специальный режим работы ядра (package mode), в котором выдаваемый текст содержит дополнительную информацию, например, указание на текущее состояние ядра или запрос на выполнение надстройкой некоторой функции (функция WindowCmd ядра), а среда разработки фактически не функционирует. Автор предлагает использовать термин «клиент GAP»для

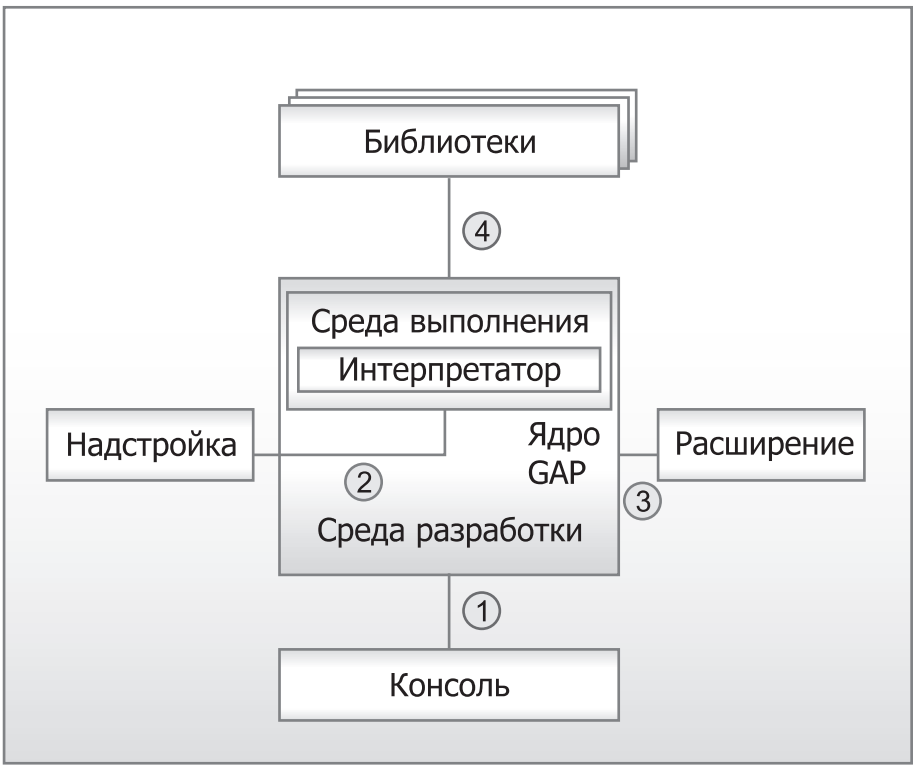

Рис. 1. Схема пакета GAP

всех программ, рассчитанных на работу с ядром GAP, считая, что надстройки являются частным случаем клиентов, обладающим следующим свойством: работа с ядром происходит через псевдотерминал (в терминах win32 – анонимный канал).

Библиотека функций GAP содержит реализации большого числа алгоритмов дискретной математики, а в библиотеке данных собраны результаты выполнения некоторых алгоритмов, реализованных в библиотеке функций, для определенных начальных данных; обе эти компоненты выполнены на языке GAP. Первое обращение к библиотеке функций происходит при запуске ядра пакета GAP, когда производится синтаксический разбор файлов библиотеки и пополнение среды выполнения описанными в них функциями и структурами данных.

Описание библиотечных функций вместе с руководством пользователя пакета и другим справочным материалом содержится в документации GAP. Она выполнена в виде электронных документов и существует в нескольких

форматах, что позволяет обращаться к ней из среды разработки и при помощи веб-браузера и других программ.

Механизмом расширения библиотеки функций GAP являются пакеты дополнений. Обязательными компонентами пакета дополнений являются библиотека функций, реализующая новые алгоритмы на языке GAP, и документация, содержащая описания функций, реализованных в библиотеке функций пакета дополнений, необязательными – библиотека данных, а в некоторых случаях еще и расширения и надстройки. Для обозначения набора функций, реализованных на языке GAP и входящих в состав библиотеки функций пакета дополнений, при помощи которых реализуется обращение к надстройке или расширению, автор предлагает использовать термин «программный интерфейс GAP пакета дополнений».

2. Недостатки пакета GAP

Как было отмечено выше, разработка пакета была начата в 1985 г., когда требования к некоторым характеристикам программного обеспечения были ниже, чем сегодня. Наследуя основные архитектурные решения, пакет GAP накопил ряд недостатков, решить которые, по замыслу автора статьи, сможет программный продукт, архитектура которого описывается в следующей главе.

2.1. Пользовательский интерфейс и интерактивность

Чтобы сделать пакет доступным пользователям как можно большего числа операционных систем, разработчики GAP избегали больших требований к программному окружению ядра; так, последняя версия ядра GAP компилируется, компонуется и работает во всех UNIX-совместимых операционных системах, под Windows с применением cygwin и на компьютерах Macintosh. Поскольку на момент создания ядра ни одна графическая библиотека для Си не получила распространения во всех перечисленных операционных системах, базисом пользовательского интерфейса среды разработки GAP является интерфейс командной строки.

Для обеспечения потребностей пользователей в графическом интерфейсе Франк Селлер в 1993 г. приступил к разработке пакета дополнений XGAP, последняя, четвертая версия которого была выпущена в 1999 г. и постоянно обновляется [2]. XGAP обеспечивает возможность осуществлять контроль над графической системой XWindow из ядра GAP и управлять ядром GAP при помощи графических примитивов. Подобно разработчикам других пакетов дополнений, разработчики XGAP старались не изменять зависимости ядра GAP от программного окружения, поэтому все функции по работе с гра-

фической системой были реализованы в самостоятельном приложении, являющемся частью XGAP и представляющем собой надстройку. (По-видимому, функция WindowCmd среды выполнения ядра GAP и режим package mode были реализованы именно для работы этого пакета дополнений.)

Надстройка запускает GAP и делает работу ядра «невидимой»для пользователя в том смысле, что пользователь не видит консольного окна встроенной среды разработки GAP. Программный интерфейс GAP рассматриваемого пакета дополнений реализует функции для работы с XWindow и обеспечивает возможность отображать некоторые объекты среды выполнения и управлять ими.

По мнению автора, в выбранной архитектуре пакета имеются два ключевых недостатка. Первый состоит в том, что надстройка XGAP базируется непосредственно на XWindow, что не позволяет использовать графическую оболочку в тех операционных системах, для которых XWindow не реализована; по этой же причине внешний вид графической оболочки довольно невыразителен. Однако можно отметить, что в рамках проекта cygnus система XWindow была портирована на операционную систему Windows, а требования пользователей к графической оболочке невысоки.

Второй, гораздо более существенный недостаток XGAP кроется в модели взаимного управления ядром GAP и надстройкой XGAP: программный интерфейс GAP пакета дополнений передает графической оболочке низкоуровневые запросы на управление графической системой, а надстройка для обработки многих событий графической системы вызывает функции библиотеки пакета дополнений, которые выполняются в ядре. Рассмотрим два случая, когда у пользователя могут возникнуть неудобства от применения такого подхода.

Пример 1. Пользователь передает в среду выполнения команду, выполнение которой сопряжено с большой вычислительной сложностью; теперь ядро GAP перейдет в состояние read лишь спустя продолжительный отрезок времени. В течение этого отрезка времени любые события графической системы, такие как щелчок мышью или перемещение окон, останутся необработанными. У пользователя возникнет ощущение, что графическая оболочка «зависла».

Пример 2. Иллюстрация представляет собой граф, некоторые части которого вычисляются по запросу пользователя. Даже несмотря на наличие пакета дополнений ParGAP (см. ниже), произведение вычислений, направленных на достраивание графа, не может происходить параллельно. Кроме того, во время вычисления пользователь не может взаимодействовать с графической оболочкой, например перемещать вершины графа по иллюстрации. Наконец, низкоуровневый характер запросов к графической оболочке XGAP со сторо-

ны ядра позволяет прорисовывать граф лишь как серию линий, окружностей и подписей к ним, хотя разумнее было бы передавать графической оболочке данные о структуре графа матрицей инцидентностей, а алгоритм отрисовки графа реализовать в графической оболочке.

2.2. Поддержка распределенных вычислений

Для программирования параллельных алгоритмов на языке GAP Джин Коперман в 1995 г. создал пакет дополнений, текущее название которого ParGAP [3]. Пакет встраивает в ядро GAP несколько низкоуровневых функций для обмена данными с другими работающими экземплярами ядра GAP, после чего поверх них реализует подмножество функций интерфейса MPI (Message Passing Interface), распространенной технологии обеспечения параллельных вычислений. Упомянутые встроенные функции реализованы через интерфейс сокетов.

Пакет ParGAP находит применение для решения трудных задач в одной защищенной интрасети, но, как следует из руководства, написанного самим автором пакета, не обеспечивает аутентификации и не позволяет управлять доступом к выстраиваемой вычислительной сети. В том же источнике отмечается, что данных о стабильной работе пакета под Windows нет.

2.3. Расширение функциональных возможностей

В связи с тем, что многочисленные расширения и надстройки GAP создаются под определенное программное окружение, их использование в несовместимом программном окружении затруднено или невозможно. В тех случаях, когда функционал, предоставляемый расширением или надстройкой, можно реализовать средствами самого GAP, проблема не стоит остро, однако зачастую это не так: использование пакетов дополнений является единственным способом обращаться из среды выполнения к тем функциям операционной системы, обращение к которым не может выполнить ядро. Можно сказать, что в этом случае расширения и надстройки являются механизмами интеграции ядра GAP с платформой операционной системы, на которой оно работает. Конечно, нельзя требовать от авторов пакетов писать расширения таким образом, чтобы они могли быть использованы в любом программном окружении, но привязка пакетов дополнений к определенному программному окружению противоречит принципам, положенным в основу группы, работающей над ядром GAP.

Стратегия непредоставления высоких требований к программному окружению, в котором производится сборка ядра, привела к тому, что для обеспечения работы заданного пакета дополнений пользователь зачастую вынужден производить установку и настройку некоторого программного продукта,

а если это программное обеспечение не работает в используемой им операционной системе – установить подходящую операционную систему или отказаться от использования пакета дополнений. Любому пользователю GAP под Windows заметно, что разработчики пакета все же ориентируются на программное окружение операционных систем класса Unix, поскольку многие пакеты дополнений не работают под Windows. Между тем за последние 10 лет широкое распространение получили языковые платформы, не зависящие от операционной системы: tcl, perl, python, java. Автору видится перспективным осуществление интеграции двух языковых сред – GAP, являющейся средством для разработки алгоритмов, и одной из перечисленных выше, богатой с точки зрения функционала. В тех же случаях, когда требуемой службы нет в интегрируемой языковой платформе, разработчики смогут реализовывать модули для соответствующей виртуальной машины; хотя такие модули будут привязаны к операционной системе, пользователями новой библиотеки станет вся аудитория пользователей интегрируемой языковой платформы.

3. Архитектура новой программной системы

Различными разработчиками было предложено несколько платформ для разработки программного обеспечения, среди которых выделяется платформа Java за счет наличия виртуальных машин для большинства операционных систем и богатства стандартной библиотеки классов. Автор предлагает архитектуру новой программной системы, в которой ядро GAP занимает место интерпретатора, лишаясь функции среды разработки, а надстройка, написанная на языке Java, интегрирует среду выполнения GAP и несколько экземпляров виртуальных машин Java. При такой интеграции отпадает необходимость во многих непереносимых пакетах дополнений, поскольку весь или почти весь нужный функционал предоставляет виртуальная машина Java.

3.1. Взаимодействие

Ключевое место в описываемой архитектуре занимает сервер, являющийся приложением Java. Сервер запускает ядро GAP и загружает в его среду выполнения специальный пакет дополнений JGAP, реализующий программный интерфейс GAP для обращения к виртуальной машине Java при посредничестве сервера.

Запуск экземпляра ядра GAP происходит при открытии нового соединения с сервером со стороны клиента или по команде администратора. Клиент, являющийся апплетом или приложением Java, проходит аутентификацию и обменивается с сервером удаленными интерфейсами: сервер предоставляет клиенту интерфейс для работы с запущенным экземпляром ядра GAP, а кли-

ент серверу – интерфейс для передачи запросов к виртуальной машине Java на стороне клиента. По запросу со стороны ядра серверу может быть поручено открыть соединение с другим работающим сервером (для этого функции пакета дополнений сообщается информация для аутентификации и сетевой адрес удаленного сервера, а также имя, под которым локальный сервер будет известен удаленному) и обменяться с ним интерфейсами для обращения к объектам на стороне друг друга и обмена командами между работающими экземплярами ядра GAP; удаленный сервер при этом оповещает свой экземпляр ядра об открытии соединения. При закрытии соединения с клиентом сервер завершает выполнение запущенного экземпляра ядра GAP (соединения с удаленными серверами также закрываются), а при закрытии соединения с удаленным сервером – уведомляет ядро о возникновении этого события. Запуск ядра GAP по команде администратора происходит для создания узлов вычислительной сети, непосредственно не взаимодействующих с клиентом.

Для идентификации виртуальной машины, к объектам которой происходит обращение из ядра, используются имена: виртуальная машина сервера идентифицируется именем server, виртуальная машина клиента – именем client, а имя удаленного сервера сообщается им самим при открытии соединения. Поскольку такая идентификация происходит в паре запущенных экземпляров ядра, имя экземпляра, запускаемого администратором, указывается самим администратором при запуске.

Механизм аутентификации определяется технологией межпроцессного взаимодействия, применяемой для удаленного вызова методов интерфейсов [4], а контроль доступа осуществляется установкой ограничений на создание объектов и определением разрешений на обмен командами между работающими экземплярами ядра GAP. Нетрудно заметить, что предлагаемая схема не допускает прямого взаимодействия клиентских виртуальных машин.

Для определенных задач допустима такая архитектура, при которой клиент не предоставляет серверу удаленного интерфейса для обращения к своей виртуальной машине Java, поскольку сама виртуальная машина может отсутствовать, ведь интерфейс для обращения к ядру GAP, предоставляемый сервером клиенту, может не зависеть от платформы. Программный интерфейс GAP пакета JGAP должен корректно обрабатывать такую ситуацию.

В предлагаемой архитектуре обращение к экземпляру ядра с запросом на выполнение некоторой команды происходит в следующих случаях:

1) пользователь вводит команду в консоли клиента;

2) на стороне клиента или сервера случается событие, которое требует для своей обработки обращения к ядру. Например, событие может быть ини-

циировано одним из контроллеров оконной составляющей клиента с целью получить от ядра информацию для отрисовки;

3) от удаленного сервера поступает запрос на выполнение некоторой команды.

Невозможность одновременного выполнения двух команд в одном экземпляре ядра GAP требует наличия очереди с приоритетами, в которую попадают все запросы на обращение к ядру и из которой они извлекаются и принимаются к исполнению в зависимости от своего приоритета. Автор предлагает алгоритм вычисления приоритетов, основанный на следующих характеристиках команды:

- отправитель команды; наименьший приоритет имеют запросы, исходящие от удаленных серверов, далее следуют запросы от пользователя и наибольший приоритет имеют обращения к ядру GAP, инициаторами которых являются модули JGAP, работающие в виртуальной машине Java на стороне клиента или локально.

- коэффициент значимости отправителя; для локального сервера этот коэффициент равняется единице, а для удаленных серверов и клиента он устанавливается администратором в учетных записях пользователей, используемых для аутентификации на сервере.

3.2. Языковая интеграция

При помощи функции WindowCmd, встроенной в среду выполнения, функции программного интерфейса GAP пакета дополнений JGAP передают серверу запросы на вызов метода некоторого объекта и считывание или запись его свойств (field), а если объект является массивом – считывание или запись его элементов по индексу. Сервер хранит вектор из объектов типа java.lang.Object и, выбрав нужный объект по его индексу, использует метод getClass для получения объекта класса java.lang.Class и далее методы getDeclaredField и getDeclaredMethod с именами свойства или метода для получения объектов типов java.lang.reflect.Field и java.lang.reflect.Method соответственно.

Программный интерфейс GAP пакета JGAP содержит функции для добавления новых объектов в репозиторий. (Сервер может передать запрос на создание объекта клиенту или удаленному серверу.) Добавляемый объект либо принадлежит одному из типов, эквивалентных встроенным типам среды выполнения GAP таким, как String и Integer, либо принадлежит одному из классов виртуальной машины, либо реализует интерфейс, определен-

ный в виртуальной машине Java, путем указания имен реализуемого интерфейса и функции-диспетчера среды выполнения, которая будет вызываться для выполнения нужного метода интерфейса; это позволит реализовывать модели для контроллеров Swing, созданных на стороне клиента, на языке GAP. Возможность реализовать интерфейс Java по его имени обеспечивает метод defineClass загрузчика классов виртуальной машины (bootstrap class loader) [5].

Обращение к свойству объекта происходит в два этапа. На первом этапе считывания значение, возвращаемое методом get класса Field, помещается в репозиторий, а номер объекта в репозитории сообщается в среду выполнения; на втором этапе производится попытка привести объект по его номеру в репозитории к одному из «примитивных»типов и передать соответствующее значение в среду выполнения. Запись свойства происходит аналогичным образом в обратном порядке: создание объекта примитивного типа в репозитории и затем присваивание его полю. При вызове метода на первом этапе создается массив объектов-аргументов, после чего происходит вызов метода с сохранением возвращаемого значения в репозиторий и далее – считывание возвращенного значения и измененных значений аргументов. При выполнении любой из перечисленных операций может возникнуть исключительная ситуация; в этом случае объект-исключение будет размещен в репозитории, а в среду выполнения будет возвращен номер этого объекта.

Нетрудно заметить, что предлагаемые механизмы языковой интеграции не включают всех возможностей языка Java, например, отсутствует возможность использовать многие операторы. Кроме того, некоторые операции, выполнение которых на языке Java было бы тривиальным, потребуют написания длинных кусков кода на языке GAP. Автор предлагает модульную архитектуру пакета дополнений JGAP, в которой установкой архива Java, содержащего реализацию специального интерфейса (далее – модуля расширенного синтаксиса), и проведением настройки сервера пользователь может добиться появления в программном интерфейсе GAP пакета дополнений JGAP функций для более простого обращения к объектам среды выполнения Java. Так, например, интересным представляется удаление из среды выполнения ядра GAP функций для работы с файловой подсистемой, запуска процессов и т. д. и реализация их средствами виртуальной машины сервера. На рис. 2 прямоугольниками с цифрами внутри обозначены работающие экземпляры GAP. Альтернативная нумерация (в «кружочках») применяется для операций и будет разъяснена далее; в тексте соответствующие блоки отмечены цифрами в круглых скобках.

При открытии соединения между клиентом и сервером (1) происходит запуск экземпляра номер 1 ядра GAP. Таким образом пользователь может при

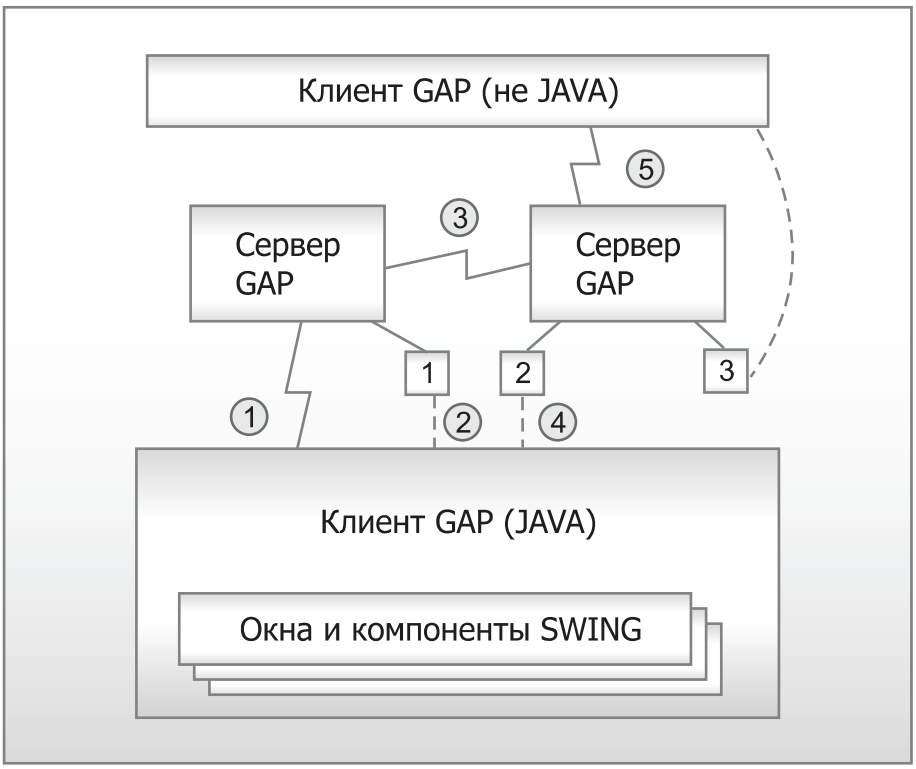

Рис. 2. Схема пакета дополнений JGAP

при помощи клиента и при посредничестве сервера работать с пакетом (2). По запросу (запрос может быть сгенерирован пользователем явно либо неявно одной из программ, запущенных пользователем) сервер открывает соединение с другим сервером (3), на котором командой администратора был запущен экземпляр ядра GAP для нужд пользователей вычислительной сети. Теперь пользователю доступны два экземпляра ядра (4). В обратном направлении от программ, запускаемых пользователем внутри ядра GAP, клиенту поступают запросы на обращения к объектом его виртуальной машины Java, в результате чего пользователь видит иллюстрации и консоль, аналогичные XGAP. Однако если соединение открывается между сервером и клиентом, написанным на любом другом языке и взаимодействующим с сервером при помощи независимой от платформы технологии межпроцессного взаимодействия (5), то отправления клиенту запросов на обращения к его объектам будут неуместны в силу различий в программных окружениях. В этом случае ядро GAP выполняет функции калькулятора.

4. Обоснование выбранного подхода

Первым и основным достоинством пакета JGAP является предоставление возможности разработчикам пакетов дополнений избегать создания расширений в тех случаях, когда производительность виритуальной машины Java является достаточной. В большей степени это замечание касается пакетов для визуализации, экспорта и импорта данных из XML и реляционных баз данных. Действительно, платформа Java, будучи востребованной в прикладном программировании, отвечает запросам разработчиков прикладных программ, функционал которых в большей степени сводится к работе с сервером баз данных, обработке данных, не требующей сложных вычислений, и взаимодействию с пользователем через оконный интерфейс.

Выше было отмечено, что при использовании программного интерфейса JGAP для работы с объектами, живущими в виртуальной машине Java, разработчик продолжает пользоваться языком GAP; это является важной особенностью пакета. Разработчик может даже не знать синтаксиса языка Java, хотя документация по этой языковой платформе будет обязательным инструментом его работы.

Графическая оболочка является неотъемлемой частью пакета JGAP. Обеспечивая обратную совместимость с пакетом XGAP, графическая оболочка JGAP предоставляет более богатый инструментарий разработчикам математических курсов и программ визуализации; часть требуемого функционала может быть реализована в виде классов Java и использована пользователями GAP наравне со стандартными классами. Не ставится под сомнение тот факт, что функционал графических библиотек Java несравнимо слабее функционала специализированных систем визуализации.

Распределенный характер модулей пакета JGAP дает пользователям GAP все преимущества клиент-серверной модели, среди которых масштабируемость (клиент может работать на слабой машине, а задача может обсчитываться на нескольких мощных машинах одновременно) и разделение на уровни (в терминах трехуровневой модели клиент JGAP играет роль уровня пользовательского интерфейса, сервер JGAP – роль сервера бизнес-логики или сервера приложений).

4.1. Ограничения интерпретатора языка

Выше было отмечено, что невозможность выполнять одновременно несколько команд в одном экземпляре GAP является существенным ограничением. Серьезность проблемы раскрывается также тем фактом, что при запуске большого числа экземпляров ядра, каждый из которых работает в отдельном процессе, сильно снижается эффективность работы системы. Автор отмечает,

что запущенный экземпляр GAP занимает в памяти 24 Мб, из которых 10 Мб занимает исполняемый код (данные получены с помощью утилиты Process Explorer [6]; использовался дистрибутив GAP версии 4.4 без дополнительных пакетов; операционная система Windows XP); резервирование 14 Мб виртуального адресного пространства для поддержания небольшого пользовательского сеанса – непозволительная роскошь.

Еще одним важным вопросом, требующим решения, является реализация потоков выполнения в ядре GAP. Для описываемой архитектуры это означает, что наборы инструкций по обращению к серверу GAP смогут выполняться в отдельном потоке, тогда как основной поток занимается поддержанием окон графической оболочки в адекватном состоянии. Выше автор предлагает компенсировать отсутствие рассматриваемого функционала такой архитектурой сервера GAP, при которой обращение к объекту виртуальной машины Java может происходить асинхронно, но поскольку одна операция будет, как правило, состоять из передачи серверу нескольких подобных запросов, предложенное временное решение не будет эффективно без применения модулей расширенного синтаксиса. Также временным решением будет косвенное (через номер в репозитории) обращение к объектам среды выполнения Java. Автор считает интересным развитие среды выполнения GAP в таком направлении, при котором привязка имени к объекту может производиться и за пределами самой среды выполнения.

В настоящее время разработчики специализированных динамических языков занимаются созданием виртуальной машины Parrot [7], предназначенной для выполнения байт-кода, в который могут компилироваться программы на языках Perl, Python, Tcl, Ruby, Scheme и многих других; продолжают развиваться виртуальные машины .NET и Java. Автору представляется перспективной реализация компилятора программ на языке GAP в байт-код одной из перечисленных виртуальных машин, поскольку реализация такого компилятора является более простой задачей, чем внесение всех упомянутых выше изменений в ядро GAP.

Литература

1. The GAP Group, GAP – Groups, Algorithms, and Programming, Version 4.4; 2005 [Электрон. ресурс]. Режим доступа: http://www.gap-system.org

2. The GAP Group, GAP – Groups, Algorithms, and Programming, Version 4.4; 2005, package XGAP [Электрон. ресурс]. Режим доступа: http://www.gap-system.org

3. The GAP Group, GAP – Groups, Algorithms, and Programming, Version 4.4; 2005, package ParGAP [Электрон. ресурс]. Режим доступа: http://www.gap-system.org

4. [Электрон. ресурс]. Режим доступа: http://www.javaworld.com/javaworld/javaqa/2000-03/03-qa-0324-ipc.html

5. [Электрон. ресурс]. Режим доступа: http://java.sun.com/docs/

6. [Электрон. ресурс]. Режим доступа: http://www.sysinternals.com/Utilities/ProcessExplorer.html

7. [Электрон. ресурс]. Режим доступа: http://www.parrotcode.org