УДК 004.75

С.А. Смирнов
Московский физико-технический институт (государственный университет)

Разработка Grid-сервиса для системы компьютерной алгебры Maxima на основе промежуточного программного обеспечения Ice

Изложены результаты работы по созданию Grid-сервиса компьютерной алгебры на основе системы Maxima (с открытым исходным кодом) средствами промежуточного программного обеспечения Ice (Internet Communications Engine). Данный сервис предоставляет пользователю простой механизм удалённого вызова операций Maxima, а также возможности создания/удаления исполняемых процессов Maxima на удалённом хосте. Статья содержит описание основных способов программного взаимодействия с системой Maxima и принципов реализации сервиса.

Ключевые слова: Maxima, grid-сервис, грид-сервис, символьные вычисления, система компьютерной алгебры, ZeroC Ice.

Maxima [1, 2] — это система работы с символьными и численными выражениями, включающая дифференцирование, интегрирование, разложение в ряд, преобразования Лапласа, обыкновенные дифференциальные уравнения, системы линейных уравнений, многочлены, множества, списки, векторы, матрицы и тензоры. Maxima производит численные расчёты высокой точности, используя точные дроби, целые числа и числа с плавающей точкой произвольной точности. Система позволяет строить графики функций и статистических данных в двух и трёх измерениях. Исходный код Maxima может компилироваться на многих системах, включая Windows, Linux и Mac OSX. На SourceForge доступны исходные коды и исполняемые файлы для Windows и Linux.

Maxima — потомок Macsyma, легендарной системы компьютерной алгебры, разработанной в начале 60-х годов XX века в MIT. Это единственная, основанная на Macsyma, система, все ещё публично доступная и, благодаря своей открытости, имеющая активное сообщество пользователей. В свое время Macsyma произвела переворот в компьютерной алгебре и оказала влияние на многие другие системы, в числе которых Maple и Mathematica.

Несмотря на почтенный возраст и наличие таких мощных конкурентов, как Mathematica, Maple и MATLAB (в котором есть Symbolic Toolbox), популярность системы не уменьшается и сообщество пользователей системы растёт, в том числе и в России.

В статье будут изложены результаты работы по созданию Grid-сервиса системы компьютерной алгебры Maxima на основе промежуточного программного обеспечения (ППО) Ice [3]. Данный сервис должен предоставить пользователю удобный (программный) механизм удалённого вызова операций Maxima, а также возможности создания/удаления экземпляров Maxima на хосте. В частности, сервис может быть применён для организации распределённых символьных вычислений, например, численных расчётов без ошибок округления.

Рассмотрим те возможности Maxima, что позволяют ей взаимодействовать с другими приложениями в интерактивном режиме. Сложность ситуации в том, что в проекте Maxima пока не документирован протокол взаимодействия с системой и отсутствует соответствующий API. Источником информации послужил анализ исходных кодов графического интерфейса wxMaxima, а также архивы рассылок проекта Maxima.

Было выявлено два способа взаимодействия с системой Maxima: на основе TCP-сокетов или с использованием механизма unnamed pipes. Начнём с первого.

Данный механизм наиболее распространён и использован в wxMaxima и мно-

гих других приложениях, взаимодействующих с Maxima. В механизме присутствуют два процесса, обязательно выполняемых на одном хосте: процесс Maxima и процесс приложения-пользователя, взаимодействующего с Maxima (например, wxMaxima). Процесс приложения-пользователя создаёт TCP-сокет и ждет входящего соединения на известный порт, например, 12345 (рис. 1). Затем запускают процесс Maxima с опцией «-s» номер порта приложения-пользователя >, например, Maxima -s 12345. При этом Maxima создаёт TCP-сокет и подключает его к localhost: 12345. После чего Maxima перенаправляет стандартные потоки ввода/вывода на созданный сокет. В результате приложение-пользователь получает сокет, соединённый с Maxima и ведущий себя как консоль Maxima. Важно отметить, что приложение-пользователь в данной схеме играет роль сервера, обслуживающего запросы процесса Maxima.

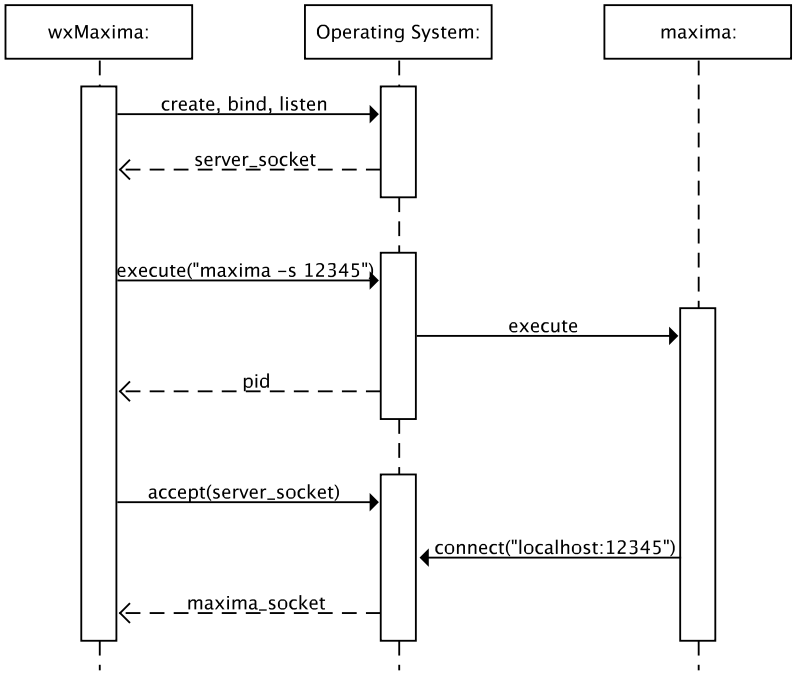

Рис. 1. UML-диаграмма последовательности запуска и установления соединения с процессом Maxima при использовании сокетов

Во втором способе процесс Maxima запускают как дочерний процесс приложения-пользователя (рис. 2). Стандартные потоки ввода/вывода Maxima открывают в приложении-пользователе как unnamed pipes. Таким образом, появляется доступ к консоли Maxima.

Произведём сравнение описанных способов взаимодействия с Maxima. Поскольку оба способа взаимодействия, начиная с момента подключения к консоли Maxima, становятся более-менее равносильными, сосредоточимся на особенностях самого процесса установления соединения. Для начала перечислим особенности подхода, основанного на сокетах.

1. Сходный API BSD сокетов присутствует в большинстве современных операционных систем.

2. Классы, обеспечивающие переносимое использование сокетов, присутствуют только в составе больших библиотек (например, в ACE).

3. Необходимы отдельные средства для запуска процесса Maxima.

Особенности использования unnamed pipes.

1. Различные платформы имеют различный API для порождения дочерних процессов и для перенаправления стандартных потоков ввода/вывода.

2. Существует небольшая, переносимая, не требующая компиляции библиотека, реализующая описанную выше функциональность (Boost.Process).

В случае сокетов проблема запуска процесса Maxima решается относительно про-

сто — использованием функции system из стандартной библиотеки языка C.

Таким образом, при использовании сокетов либо будет добавлена зависимость от большой библиотеки, либо необходимо будет разработать и протестировать дополнительный программный код, обеспечивающий запуск и установление соединения с процессом Maxima. При этом перенесение такого кода на другие платформы будет осуществляться относительно просто благодаря схожему API сокетов. В случае использования unnamed pipes количество нового кода минимально и переносимость обеспечена автоматически, однако будет добавлена зависимость от библиотеки Boost.Process.

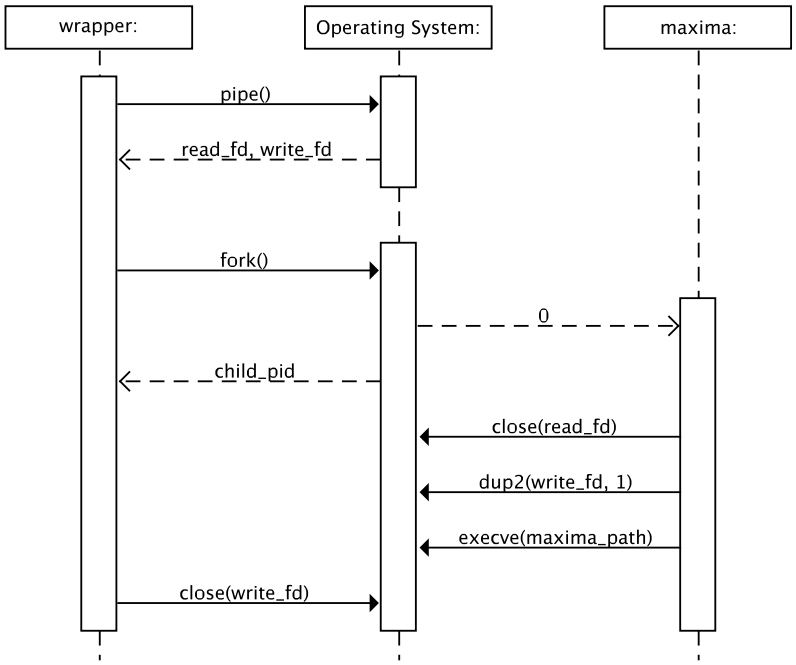

Рис. 2. UML-диаграмма последовательности запуска и установления соединения с процессом Maxima при использовании unnamed pipes

В данной работе был выбран второй способ взаимодействия, так как он обеспечивает работу с процессом Maxima с меньшими трудозатратами, добавляя зависимость от сторонней библиотеки, не требующей компиляции.

После подключения к консоли Maxima можно перейти к задаче взаимодействия с системой. Протокол общения через консоль с Maxima довольно прост: в ответ на запрос со стороны Maxima пользователь вводит выражение. Maxima выдаёт некоторый результат. Далее цикл повторяется. Иногда, если пользователь вводит определённые выражения, Maxima ведёт себя нестандартно: не выдаёт ответ или не печатает запрос на следующее действие пользователя. Судя по поведению существующих приложений, взаимодействующих с Maxima, это ошибка текущей версии Maxima. Таким образом, если пользователь придерживается некоторого недокументированного формата выражений, то протокол взаимодействия с Maxima может быть охарактеризован как запрос-ответ.

Автоматическая обработка консольного вывода Maxima довольно сложна: не просто выделить общий вид вывода Maxima, представляющий собой результат, или сообщение, или запрос ввода. Согласно doc/implementation/external-interface.txt из дистрибутива Maxima, в системе присутствует ряд переменных, управляющих выделением значимых фрагментов консольного вывода, что позволяет значительно упростить автоматическую обработку вывода Maxima. Перечислим использованные нами переменные и их смысл:

1) *prompt-prefix* — строка, печатаемая перед каждым запросом на ввод;

2) *prompt-suffix* — строка, печатаемая после каждого запроса на ввод;

3) *alt-display2d* — функция, заменяющая стандартную при печати выражений Maxima в 2-мерном режиме.

Установка этих переменных происходит в файле, написанном на языке Common Lisp, путь к которому указывают при помощи опции $-p <$ Путь к файлу $>$. Выполнение содержимого данного файла происходит на этапе загрузки Maxima.

В этой статье описывается второй вариант сервиса: ранее уже был реализован аналогичный сервис, обеспечивающий доступ к ресурсу Maxima для первой версии системы распределённых вычислений IARnet (IARnet1) [4, 5]. При этом возник ряд трудностей, связанных с недостатками IARnet1, в частности:

1) весьма примитивный язык описания интерфейсов;

2) необходимость избыточного преобразования указателей;

3) необходимость использовать упрощённый интерфейс вызова операций;

4) отсутствие статической проверки типов.

Переход на ППО Ice решает эти проблемы. Кроме того, Ice был выбран в качестве ППО для новой версии IARnet — IARnet2 [6], что позволит легко перенести существующий сервис на платформу IARnet2.

Перейдём теперь к программной реализации, начав с описания интерфейса сервиса Maxima для Ice, включающего два интерфейса: объекта фабрики и объекта экземпляра Maxima. Изначально пользователь обладает лишь прокси объекта фабрики (создаваемого по известной ссылке на Ice-объект фабрики). По запросу пользователя (launchMaxima) объект фабрики создаёт отдельный экземпляр Maxima и «заворачивает» его в соответствующий объект, прокси которого возвращается пользователю. Объект экземпляра Maxima предоставляет доступ к системе Maxima по принципу запрос-ответ. Описанная схема использования сервиса проиллюстрирована на рис. 3. Описание интерфейса сервиса на языке Slice (принятом в Ice) выглядит следующим образом:

```
module Maxima {
exception MaximaError {
string reason;
};
interface MaximaResource {
string executeCommand (string command)
throws MaximaError;
string executeInteractive (string
command) throws MaximaError;
void destroy ();
};
interface MaximaFactory {
MaximaResource *launchMaxima () throws
MaximaError;
};
};
```

Опишем операции интерфейса объекта экземпляра Maxima (MaximaResource):

1) executeCommand передает свой аргумент как следующую команду процессу Maxima. Возвращает последний из результатов, выданных Maxima. При ошибке выбрасывается исключение;

2) executeInteractive передает свой аргумент как следующую команду процессу Maxima. Возвращает весь ответ Maxima на данную команду, в том числе и ошибки;

3) destroy уничтожает данный объект вместе с экземпляром Maxima.

В процессе реализации сервиса весь код, отвечающий за взаимодействие с Maxima, был выделен в самостоятельный модуль, что позволило отделить логику, отвечающую за взаимодействие с процессом Maxima, от кода, обеспечивающего интеграцию в распределенную систему. Перечислим особенности реализации модуля, отвечающего за взаимодействие с Maxima.

Как уже отмечалось, при разработке данного модуля был избран способ взаимодействия с консолью Maxima, основанный на использовании unnamed pipes. В основу реализации легла модифицированная нами версия библиотеки Boost.Process [7]. Для упрощения обработки консольного вывода была применена опция $-p$ запуска Maxima с соответствующим файлом (maximag-disp.lisp). В данном файле вывод Maxima был сконфигурирован таким образом, чтобы выделить запросы на ввод, поступающие от Maxima. Вывод был настроен так, чтобы выражения-результаты выделялись особым образом и показывались в одномерном режиме, то есть были подготовлены к новой передаче в Maxima. Разбор вывода реализован на основе регулярных выражений, предоставляемых библиотекой Boost.Regex [7].

Перейдём теперь к вопросам реализации модуля, отвечающего за интеграцию Maxima в Ice. Данный модуль включает специальные программные компонен-

ты (servants), ответственные за реализацию объектов фабрик и объектов экземпляров. Также модуль содержит реализацию IceBox-сервиса. Последний отвечает за создание отдельного объектного адаптера Ice, а также позволяет инкапсулировать весь сервис Maxima в отдельную динамически загружаемую библиотеку. Объект фабрики создаёт объекты экземпляров Maxima путём простого создания экземпляра соответствующего класса реализации и его активации в объектном адаптере. Всё взаимодействие с Maxima в объекте фабрики и в объекте экземпляра сведено к вызову соответствующих операций объектов из первого модуля.

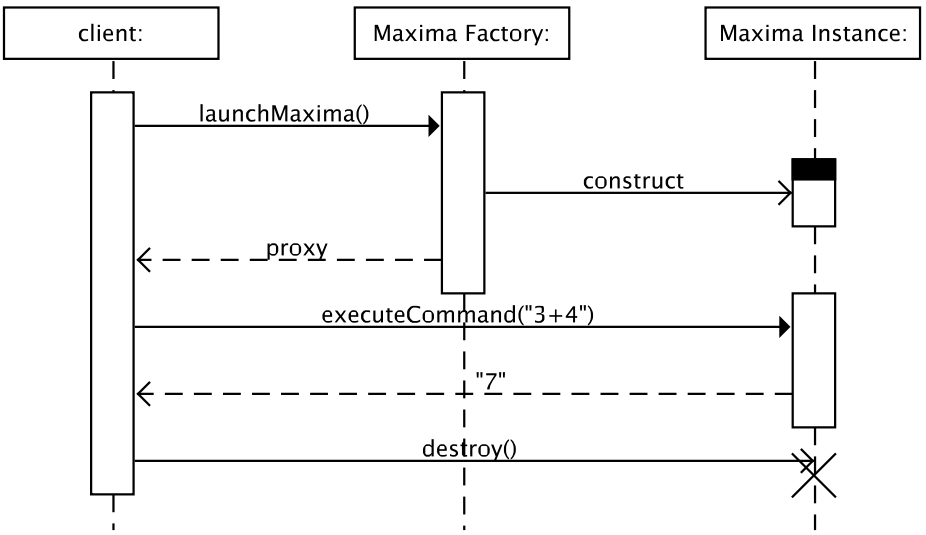

Рис. 3. UML-диаграмма последовательности использования Grid-сервиса системы Maxima

В результате проведения данной работы был реализован сервис системы компьютерной алгебры Maxima для Ice, удовлетворяющий ряду требований:

1. Организация интерактивного взаимодействия с Maxima по схеме запрос-ответ.

2. Приведение вывода Maxima к виду, допускающему автоматическую обработку.

3. Каждому экземпляру Maxima должен соответствовать отдельный Ice-объект.

4. Наличие объекта фабрики, обеспечивающего динамическое создание и получение экземпляров Maxima.

Благодаря технологиии Ice созданный сервис может быть использован программным образом из приложений на всех языках высокого уровня, поддерживаемых данным инструментарием (C++, Java, Python, C#, Ruby и т. п.).

В будущем планируется перенос сервиса на платформу IARnet2, а также ряд других улучшений: создание центральной фабрики (реестра фабрик), передача файлов между экземплярами Maxima, исправление ошибок, а также возможность прерывания выполнения команды.

Программный код описанного в статье Grid-сервиса можно загрузить с сайта http://dcs.isa. *ru/* ∼ *ssmir*

Литература

1. *Житников В.* Компьютеры, математика и свобода // Компьютерра. — 2006. — № 16 (636). — С. 40–43.

2. *Тарнавский Т.* Maxima: максимум свободы символьных вычислений (цикл статей) // LinuxFormat. — 2006. — № 81–86.

3. *Henning M.* A New Approach to Object-Oriented Middleware // IEEE Internet Computing. — 2004. — V. 8, N. 1. — P. 66–75.

4. *Емельянов С.В., Афанасьев А.П., Волошинов В.В.* [*и др.*]. Реализация Grid-вычислений в среде IARnet // Информационные технологии и вычислительные системы. — 2005. — № 2. — С. 61–75.

5. *Волошинов В.В., Естехин О.С., Сухорослов О.В.* Архитектура и принципы

реализации системы IARnet // Проблемы вычислений в распределенной среде: Модели обработки и представления данных. Динамические системы // Труды ИСА РАН. — 2005. — Т. 14. — С. 13–25.

6. *Sukhoroslov O.V.* On using Ice middleware in the IARnet framework // XXI International Symposium on Nuclear Electronics & Computing (NEC'2007): Abstracts. — 2007. — P. 48.

7. *Karlsson B.* Beyond the C++ Standard Library: An Introduction to Boost. — Addison-Wesley Professional, 2006.

Поступила в редакцию 28.12.2007.