Трансляция Акторного Пролога в Джаву

Алексей А. Морозов

morozov@cplire.ru

Институт радиотехники и электроники им. В. А. Котельникова РАН

О чём идёт речь?

Свойства логического языка:

- Используется **рекурсия** (циклов нет).
- Осуществляется **откат** (бэктрэкинг) программы (в том числе, вместо ветвлений).
- Подпрограммы могут иметь **несколько тел** (вместо одного).

Свойства императивного языка:

- Используются **циклы**, рекурсия может приводить к переполнению стека.
- Используются **ветвления**.

Зачем это нужно?

Реализация логического языка:

Система логического программирования
Механизм логического вывода
Виртуальная машина
Система распределения памяти
Операционная система
Железо

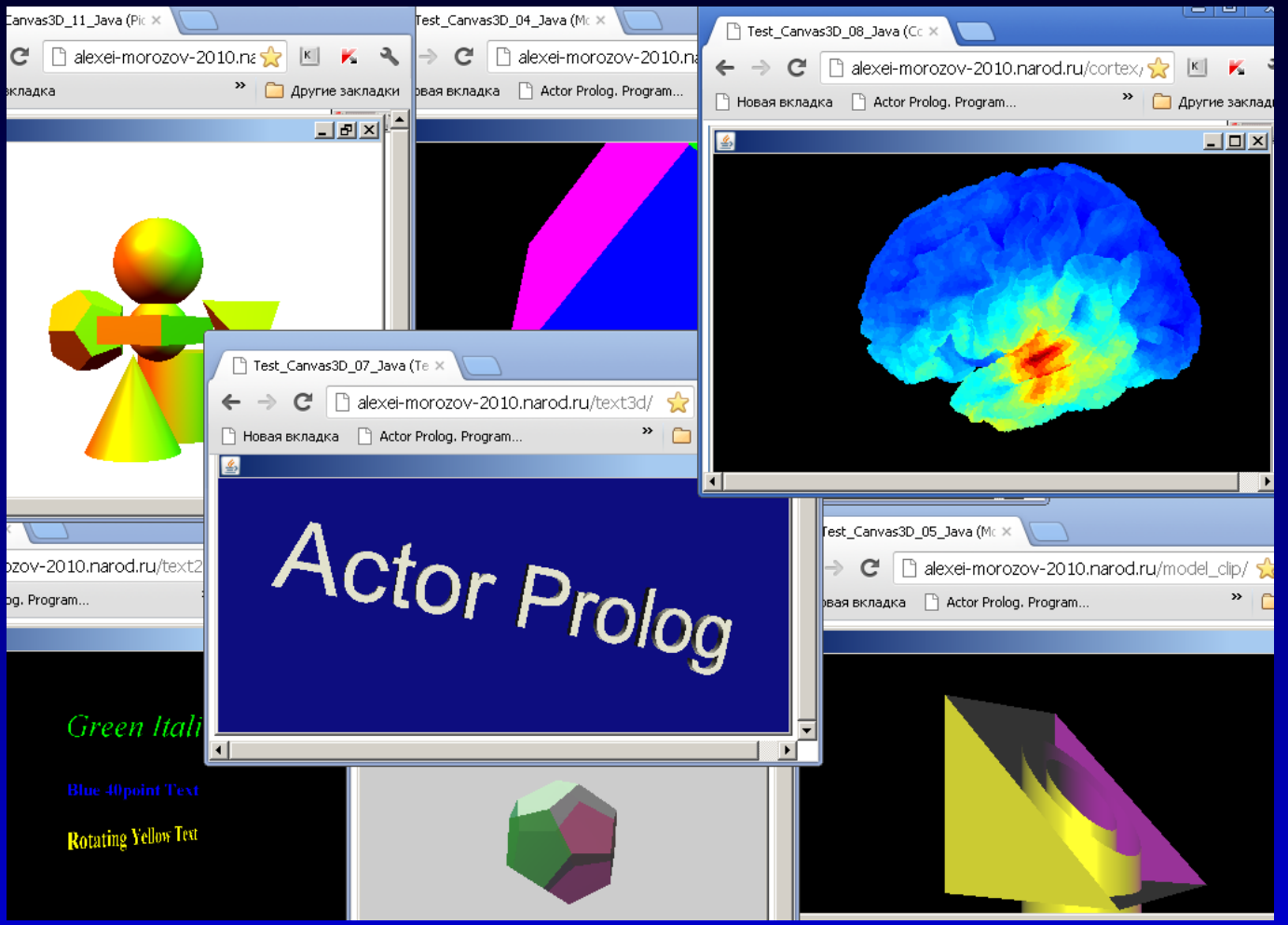
Созрели условия, при которых целесообразнее взять готовую виртуальную машину общего назначения, чем развивать специализированную виртуальную машину для логического вывода.

Скорость кода

Test	**Java**	**EXE**
NREV	34,807,018 lips	54,385,965 lips
CRYPT	8.25 ms	4.12 ms
CRYPT[!]	5.282 ms	4.16 ms
DERIV	0.085265 ms	0.00335 ms
POLY_10	19.25 ms	3.42 ms
PRIMES	0.102350 ms	0.172 ms
QSORT	0.113750 ms	0.0139 ms
QUEENS	35.562 ms	2.42 ms
QUERY	4.5984 ms	0.308 ms
TAK	10.64 ms	1.55 ms

Pentium Q9450, 2.67 GHz, 3.25 GB)

Примеры программ



Сейчас компьютеры уже достаточно быстрые, чтобы использовать такой подход на практике.

Преимущества и недостатки

Преимущества подхода:

- **Платформо-независимость** программ.
- **Стабильность** работы программ.
- **Безопасность** кода.
- Можно **использовать** все средства Джавы, в том числе графические.
- Возможность **развиваться** вместе с Джавой.

Недостатки подхода:

- **Скорость** исполнения кода меньше.
- Сложно **оптимизировать**.
- **Зависимость** от JVM.

Общая схема трансляции в Джаву

1. **Лексический** и **синтаксический** анализ программы.
2. Анализ связей между **классами** программы.
3. Проверка **доменов** аргументов предикатов, а также слотов классов.
4. Проверка **детерминированности** предикатов.
5. Глобальный **потоковый анализ** программы.
6. Генерация исходных файлов на **Джаве**.
7. Трансляция **Джава**-программы.

Базовые схемы трансляции

1. Трансляция **императивных** предикатов (императивные предикаты являются детерминированными, и при этом их выполнение всегда заканчивается успехом).
2. Трансляция **детерминированных** предикатов (детерминированные предикаты никогда не создают точек возврата).
3. Трансляция **недетерминированных** предикатов (общий случай, наиболее сложный).

Проблема не столько в том, чтобы оттранслировать, а в том, чтобы **проверить свойства** предикатов (подпрограмм).

Императивные предикаты

```
goal:-
        p.
p:-
        q.
q:-
        writeln("Hi!").

public void impProcP_s617_0(ChoisePoint iX) {
        impProcQ_s618_0(iX);
}
public void impProcQ_s618_0(ChoisePoint iX) {
        impProcWriteln_s193_1_i1(
                iX,new PrologString("Hi!"));
}
```

Анализ детерминированности

1. Анализ детерминированности отдельных предложений (возможность завершения успехом или неудачей, гарантированное завершение успехом или неудачей и др.).
2. Вычисление минимальных и максимальных ограничений, налагаемых предложением на аргументы (учитывая равенства).
3. Проверка взаимоисключаемости и взаимодополняемости предложений.

```
append([],L)= L.
append([H|R1],L2)= [H|?append(R1,L2)].
```

Детерминированные предикаты

```
goal:-
        p.
p:-
        q.
q:-
        writeln("Hi!").
```

```
public void detProcP_s617_0(ChoisePoint iX)
          throws Backtracking {
     detProcQ_s618_0(iX);
}
```

Откат реализован с помощью исключений.
Возможна оптимизация хвостовой рекурсии.

Недетерминированные предикаты

```
class NondetProcP_s617_0 extends Continuation {
    private Continuation c1;
    NondetProcP_s617_0(Continuation aC) {
        c0= aC;
    }
    public void execute(ChoisePoint iX)
                throws Backtracking {
        c1= new NondetProcQ_s618_0(c0);
        c1.execute(iX);
    }
}
```

Используются континуаторы. Возможна оптимизация хвостовой рекурсии.

Некоторые дополнительные задачи

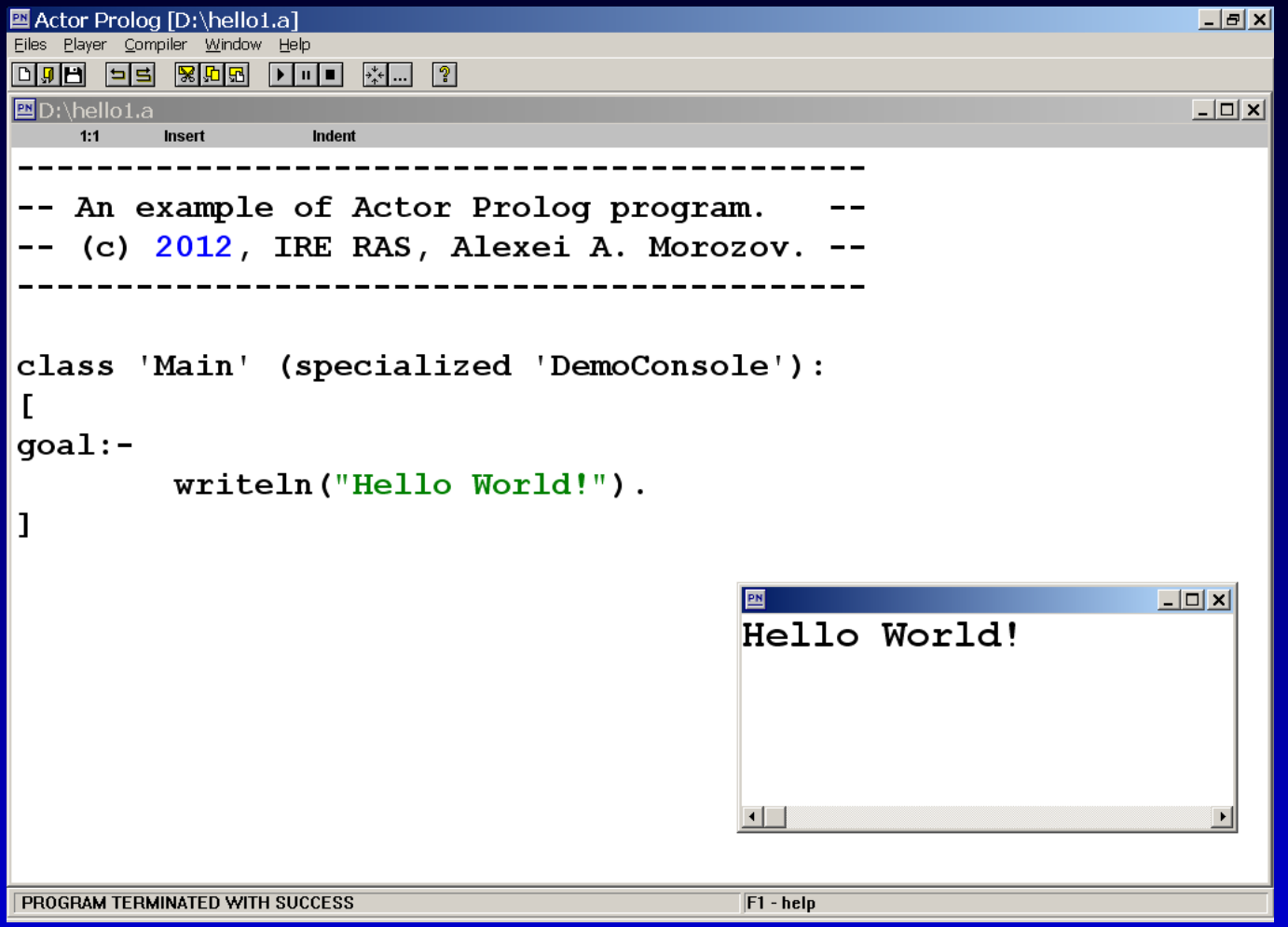
1. Реализация **унификации** аргументов подпрограмм. Различаются **основные**, **ссылочные** и **смешанные** типы данных.
2. Реализация **отсечений**.
3. Реализация **объектно-ориентированных** средств **Акторного Пролога**.
4. Реализация **параллельных процессов Акторного Пролога** средствами **Джавы**.
5. Обработка **исключительных ситуаций** и **ошибок времени исполнения**.
6. Реализация **предопределённых классов Акторного Пролога** на **Джаве**.

Скорость кода

Test	**Java**	**EXE**
NREV	34,807,018 lips	54,385,965 lips
CRYPT	8.25 ms	4.12 ms
CRYPT[!]	5.282 ms	4.16 ms
DERIV	0.085265 ms	0.00335 ms
POLY_10	19.25 ms	3.42 ms
PRIMES	0.102350 ms	0.172 ms
QSORT	0.113750 ms	0.0139 ms
QUEENS	35.562 ms	2.42 ms
QUERY	4.5984 ms	0.308 ms
TAK	10.64 ms	1.55 ms

Pentium Q9450, 2.67 GHz, 3.25 GB)

Система программирования на языке Акторный Пролог



Система поддерживает режим интерпретации, а также трансляцию в **Джаву** (**JDK7**) и в **EXE**-код.

Спасибо за внимание!

Алексей А. Морозов

morozov@cplire.ru

Институт радиотехники и электроники
им. В. А. Котельникова РАН
Моховая 11, Москва 125009

http://www.cplire.ru/Lab144/

Приложения

Краткая история проекта «Акторный Пролог»

- **1989–1991 гг.** Разработка средств имитации **логики второго порядка** (доказательное программирование, Д. Грис, Э. В. Дейкстра, синтез алгоритмов и т.п.).
  Интерпретатор логического языка, поддерживающего **«недоопределённые множества»**.
  Первые попытки реализовать идею **визуального логического программирования**.
  Идея объединения **логического** и **объектно-ориентированного** программирования.

Краткая история проекта «Акторный Пролог»

- **1991–1994 гг.** Исследование (математической) проблемы логического программирования системы, работающей в **динамическом внешнем окружении**. Связь с проблемой **разрушающего присваивания** в логическом программировании. Идея **повторных доказательств** в логическом программировании (**«логические акторы»**). Статья в журнале «Программирование» (1994, N5).
- **1996 г.** Публикация первой версии определения **Акторного Пролога**.

Краткая история проекта «Акторный Пролог»

- **1998 г.** Эксперименты с объединением **объектно-ориентированного** логического программирования с **функциональными диаграммами** (**SADT**, они же **IDEF0**) легли в основу моей диссертации (защита в ИСП РАН).
- **1999–2007 гг.** Эксперименты с объединением логического программирования с **веб-программированием** (логические агенты **Интернет**).

Краткая история проекта «Акторный Пролог»

- **2002 г.** Опубликована первая версия определения **параллельного Акторного Пролога**. Эксперименты с явным описанием параллельных процессов в **Акторном Прологе** продолжаются.
- **2003 г.** Доклад на конференции **ICLP** (Мумбаи, Индия).
- **2007 г.** Бета-версия первого **компилятора Акторного Пролога** (в **EXE**-код, под Виндами). Доклад на конференции **ICLP** (Порто, Португалия).
- **2008 г.** Начало экспериментов с трансляцией **Акторного Пролога** в **Джаву**.

Примеры решения задач на Акторном Прологе

1. Исследование асинхронных параллельных алгоритмов. **Логическая модель асинхронных параллельных вычислений**.
2. Сбор и анализ информации в Интернет. **3D-визуализация** данных (**VRML**).
3. **Анимация функциональных диаграмм** (**SADT**-диаграммы служат как инструмент **визуального программирования** и потом как **пользовательский интерфейс**).
4. Обработка текстов, генерация документов **LaTeX**, **HTML**, **VRML**).
5. Эксперименты с **трёхмерной графикой** (**Java3D**).

Модель асинхронных параллельных вычислений

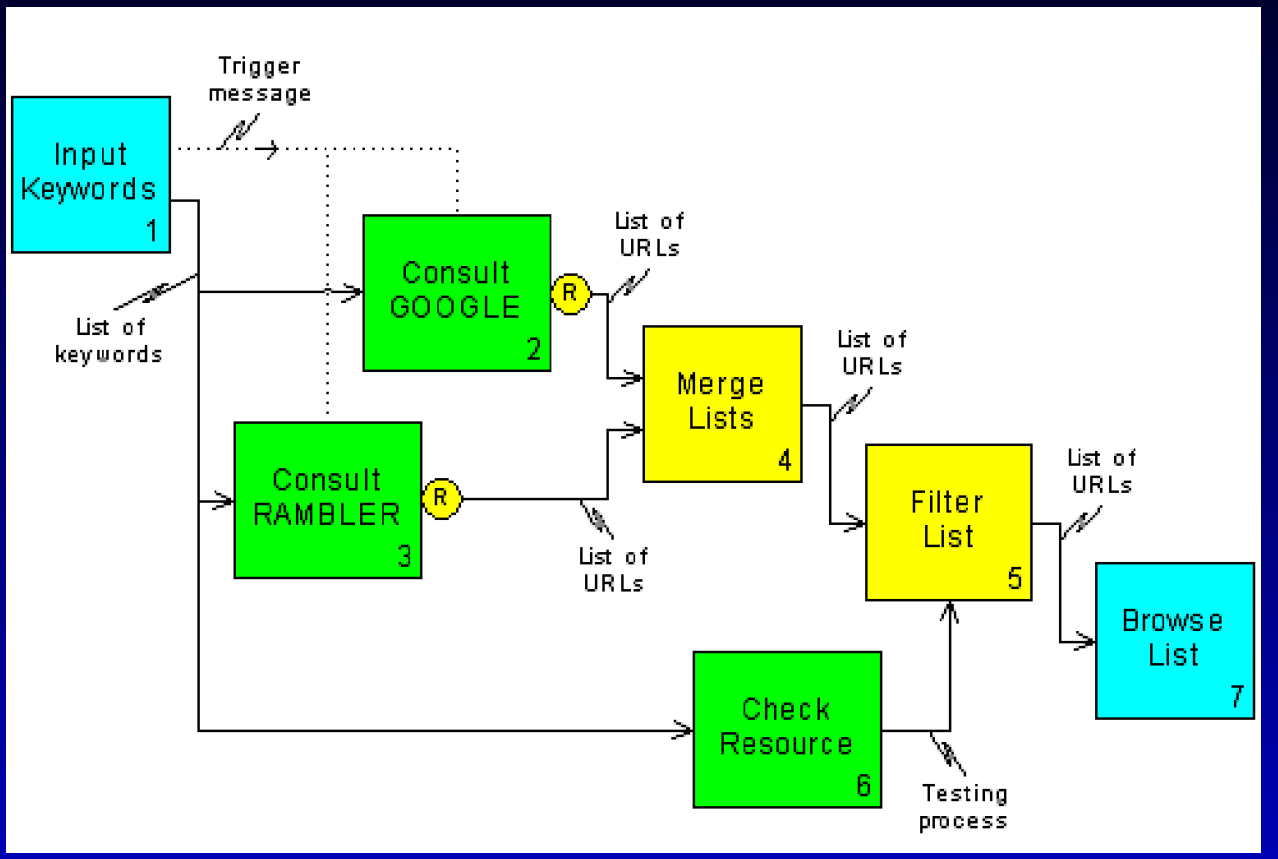
- Программа на **Акторном Прологе** состоит из процессов. **Процесс** — экземпляр класса, исполняемый параллельно.
- Взаимодействие процессов основано на идее **опережающих вычислений**.
- Наличие у логического языка теоретико-множественной семантики даёт возможность **отказаться от синхронизации процессов** в ходе их взаимодействия.
- Если исходные данные для некоторого процесса изменились, **проведённые ранее вычисления модифицируются**.

Обмен данными между процессами

В **Акторном Прологе** реализовано несколько механизмов **асинхронной** передачи данных между параллельными процессами:

1. Так называемые **прямые сообщения**. Асинхронный вызов предиката из одного процесса в другом.
2. **Потоковые сообщения** соответствуют передаче данных через общие переменные процессов.
3. **«Резиденты»** — механизм непрерывного слежения за состоянием процесса. Все ответы заданного предиката собираются в список и посылаются другому процессу.

Пример параллельной программы



Это схема потоков данных некоторого Веб-агента. Использованы прямые и потоковые сообщения, а также резиденты (R).

Пример программы на Акторном Прологе

```
class ’_Sender’ (specialized ’Dialog’):
entry_o1;
value_o1;
identifier = “Control”;
[
goal:–!.
]
```

```
dialog “Control” (
        “Sender control panel”,
        …
                radiobuttons(value_o1)
        …
end_of_dialog
```

Пример программы на Акторном Прологе

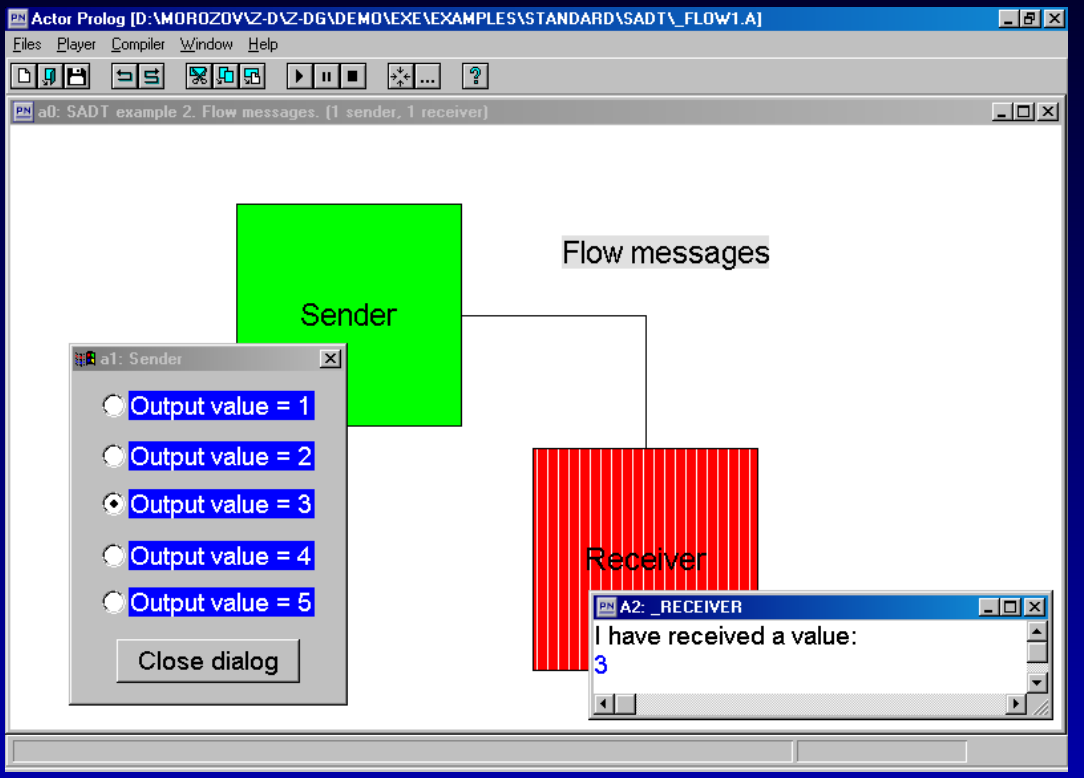
```
class ’_Receiver’ (specialized ’Alpha’):
entry_c1;
value_c1;
con;
[
goal:–!,
        con ? writeln(
                “I have received a value:”),
        con ? writeln(value_c1),
        check(value_c1).
check(#):–!.
check(Value):–
        even(Value).
]
```

Пример программы на Акторном Прологе

Идея объединения **объектно-ориентированного логического программирования** и **функционального моделирования** (**SADT**).
Программа состоит из четырёх текстовых файлов:

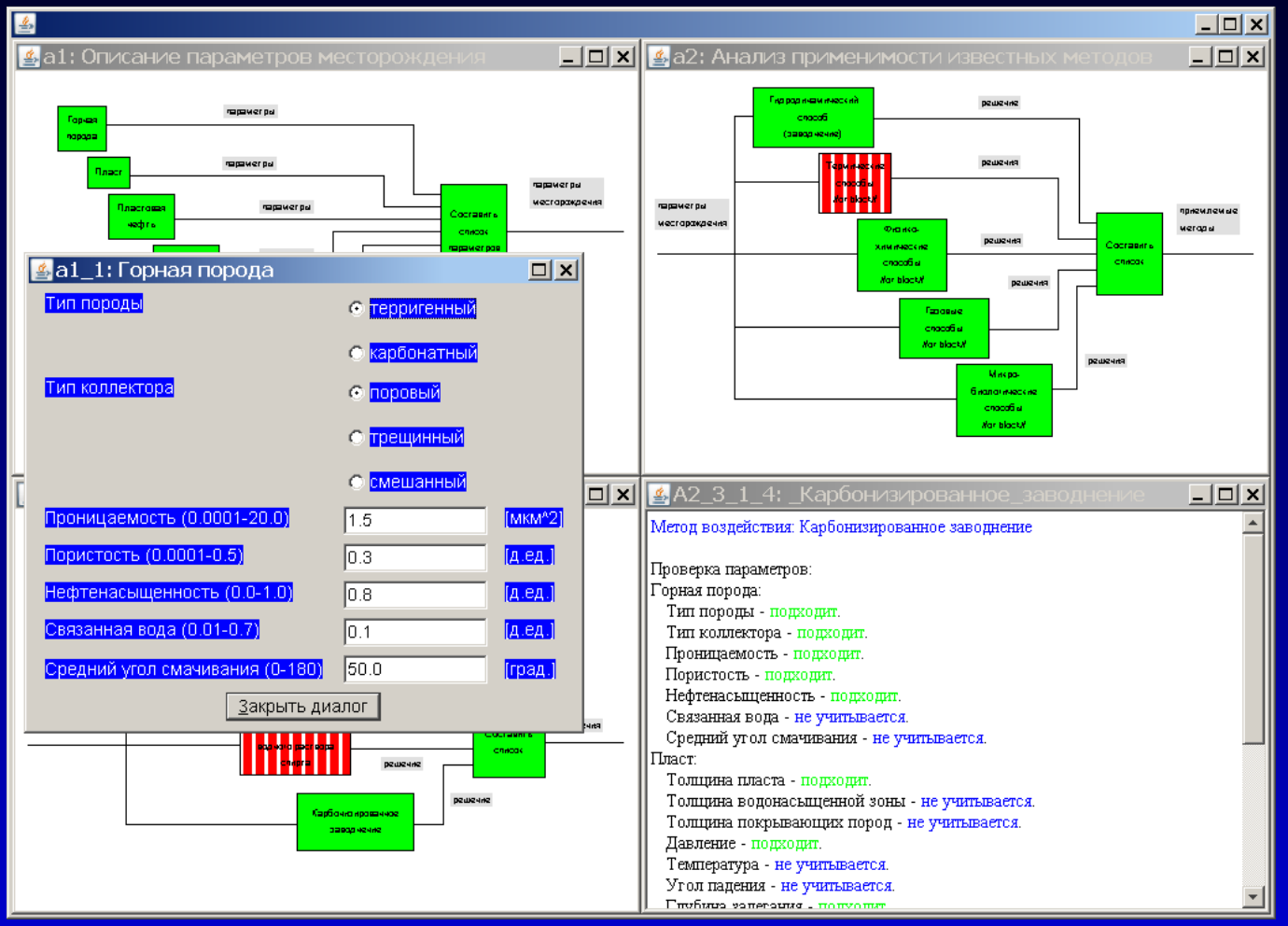
1. **Исходные файлы**, рассмотренные ранее.
2. Дополнительный исходный файл, в котором описываются **домены и предикаты** программы (не показано).
3. Определение **SADT**-диаграммы в так называемой **IDL**-нотации (текстовое описание).
4. Описание **диалоговых окон** в специальной текстовой нотации.

Пример программы на Акторном Прологе



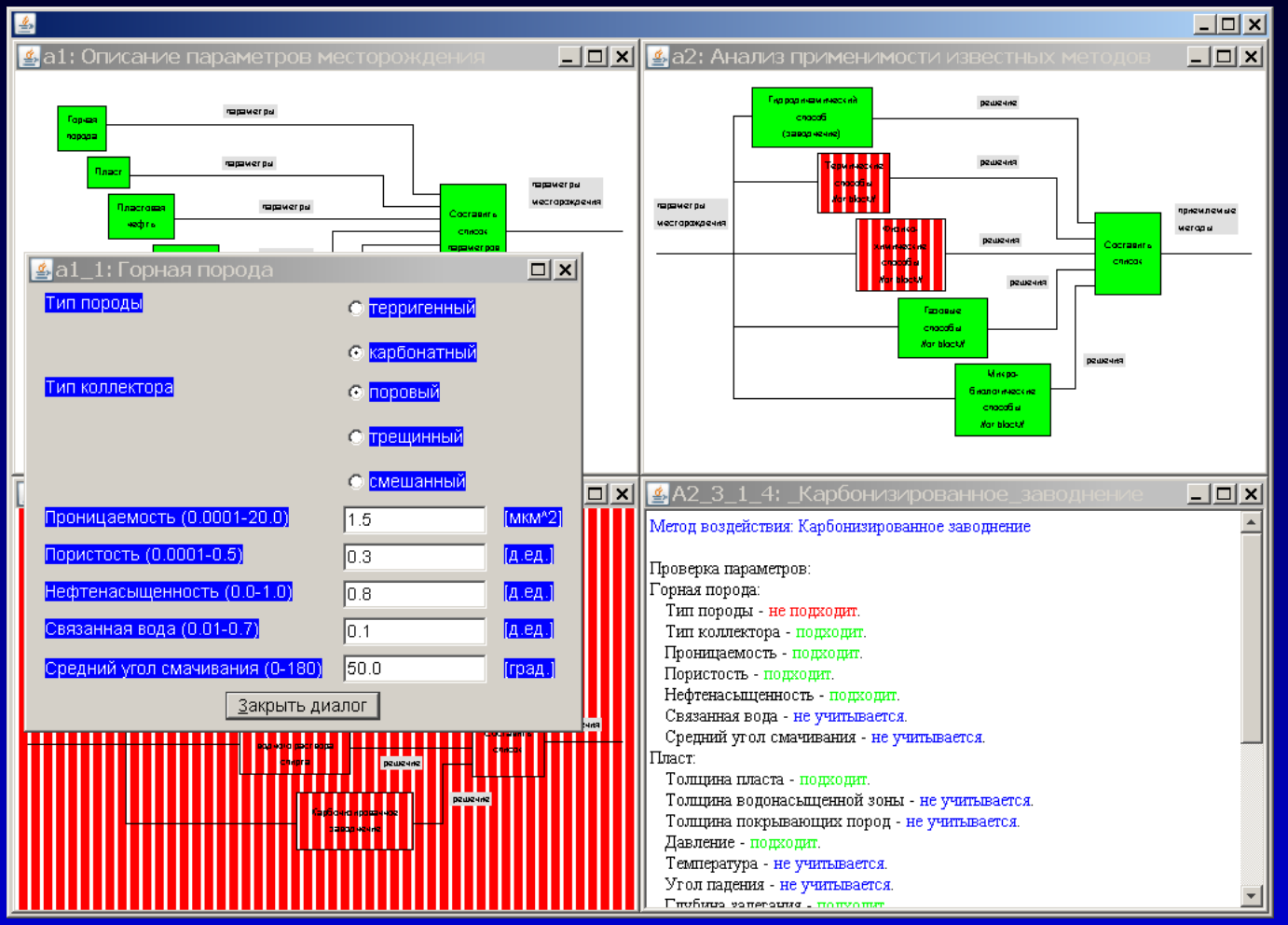
Блок **Sender** реализован с помощью процесса, посылающего потоковые сообщения, а блок **Receiver** — с помощью процесса, принимающего эти сообщения.

Пример программы на Акторном Прологе



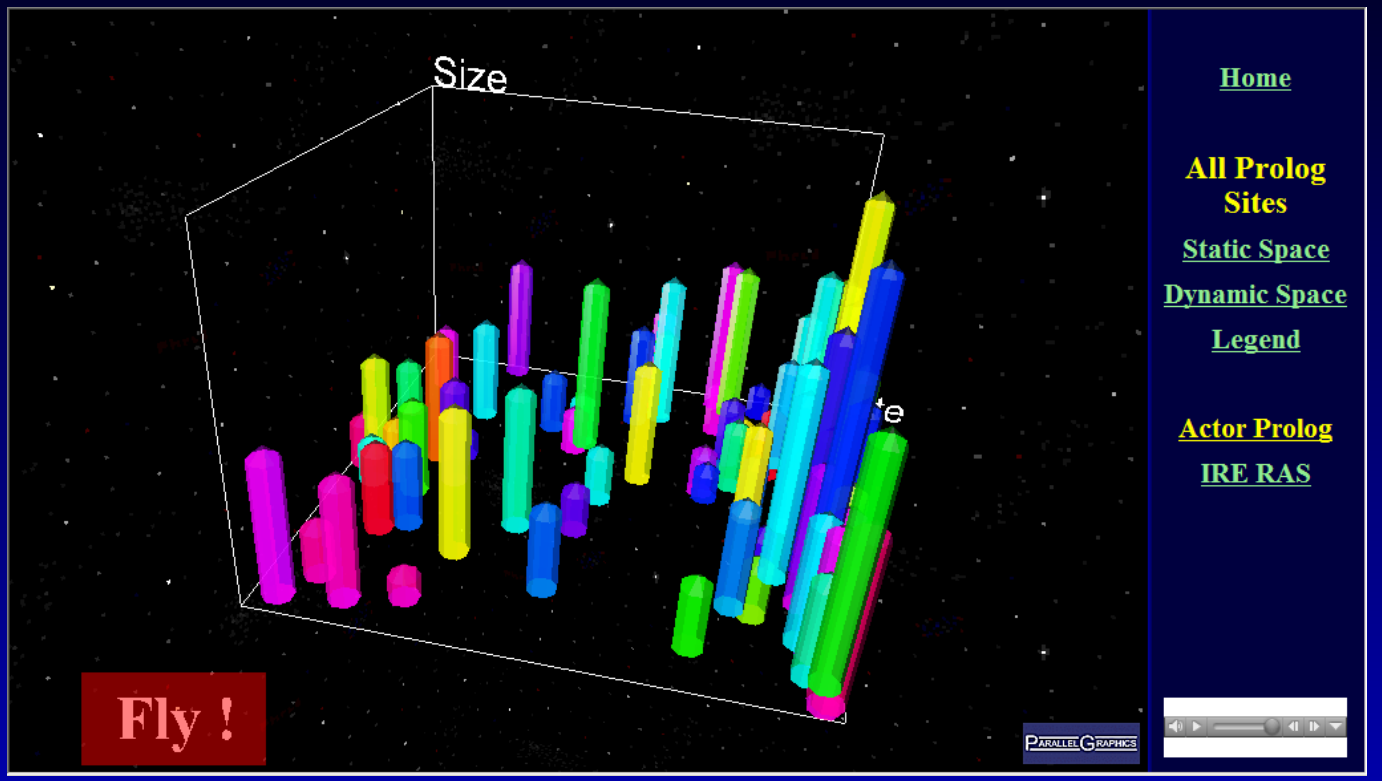
Визуальная экспертная система для выбора метода добычи нефти.

Пример программы на Акторном Прологе



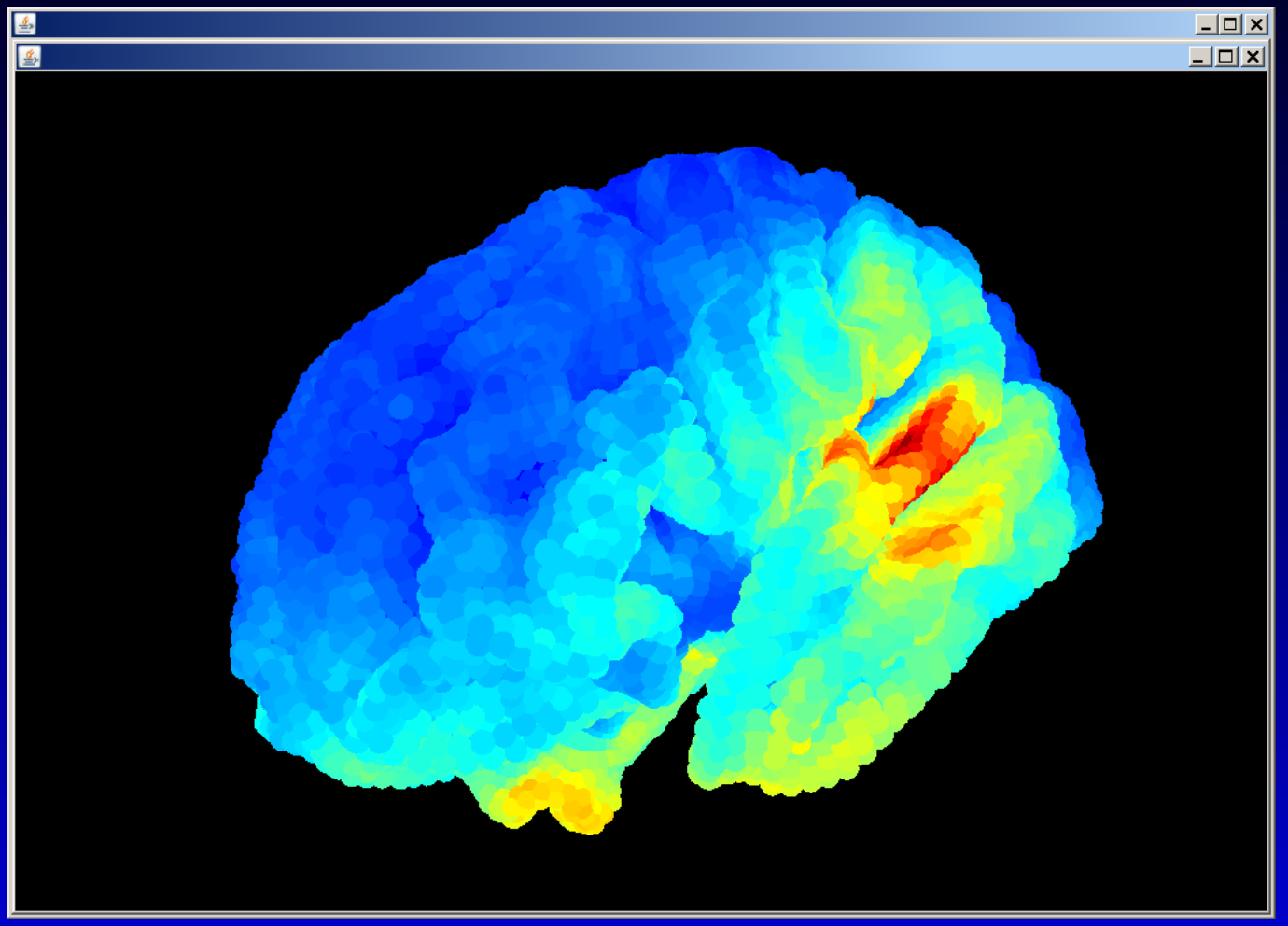
Визуальная экспертная система для выбора метода добычи нефти.

Генерация документов с помощью Акторного Пролога



3D-визуализация данных, собранных Веб-агентом (**VRML**).

3D-визуализация с помощью Актор-ного Пролога

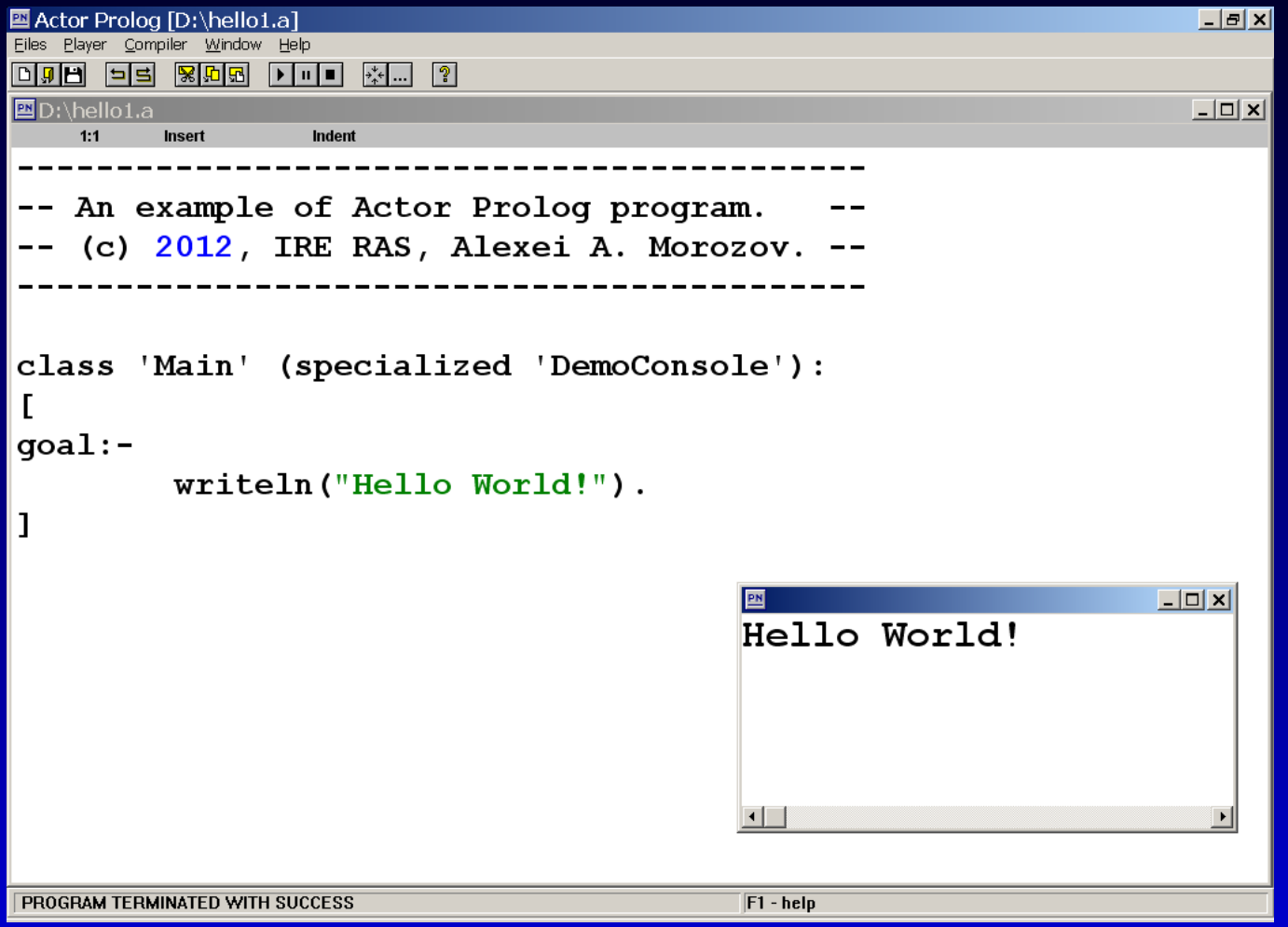


3D-визуализация результатов нейрофизиологического эксперимента (**Java3D**).

Что у меня есть интересного сегодня?

1. Первая версия **транслятора Акторного Пролога в Джаву** работает. Библиотека предопределённых классов **Акторного Пролога** воспроизведена на **Джаве** (**JDK7**).
2. Возможность **расширять систему**, просто добавляя новые классы на **Джаве**.
3. Всё это работает значительно **надёжнее**, чем компиляция в **EXE**-код. **Настоящий параллелизм**. **Джава** спасёт нас от Виндов. Можно создавать серьёзные приложения.

Система программирования на языке Акторный Пролог



Система поддерживает режим интерпретации, а также трансляцию в **Джаву** (**JDK7**) и в **EXE**-код.

Над чем я работаю сейчас?

1. Новые решения проблемы **создания объектов** в логическом программировании. Язык расширен новыми средствами (динамического) создания объектов.
2. Начал писать интерфейс с **Java3D**. Реализовано создание простых 3D-сцен.
3. Развитие **учебного курса** на базе **Акторного Пролога** и, в частности, учебной системы программирования на **Акторном Прологе** (**МГППУ**).

Возможные направления сотрудничества

1. Разработка и реализация **методов компиляции** объектно-ориентированного логического языка. В будущем — перенос исходников **Акторного Пролога** с **Вижуал Пролога** на **Акторный Пролог**.
2. Разработка новых классов на **Джаве** для подключения к **Акторному Прологу**. Примеры: эксперименты с **Java3D**, реализация **Интернет-протоколов** различного уровня, взаимодействие с **базами данных**, реализация **двухмерной графики**, эксперименты с **пользовательским интерфейсом**.
3. Реализация студенческих проектов на **Акторном Прологе**.