ОНТОЛОГИЧЕСКОЕ МОДЕЛИРОВАНИЕ ГРИД-ПРИЛОЖЕНИЙ

БАБОШИН А. А.

УДК 004.75
Бабошин А. А. **Онтологическое моделирование ГРИД-приложений.**
Аннотация. В статье описывается способ построения онтологического описания ГРИД-приложений на примере построения онтологического описания диаграмм бизнес-процессов.
Ключевые слова: онтология, ГРИД, концептуальная модель, бизнес-процесс.

UDC 004.75
Baboshin A. A. **Ontology modeling of grid-applications**.
Abstract. It describes a method of constructing ontology description of grid-applications on the analogy with constructing of ontology description of business process diagrams.
Keywords: ontology, grid, denote, business process.

1. Введение. В современных проектах для автоматизации разработки ГРИД-приложений присутствует тенденция предоставления пользователю возможностей визуального проектирование ГРИД-сценариев с последующим их выполнением на ГРИД-узле. То есть, конечному пользователю нет необходимости разрабатывать приложения на каком-либо языке программирования — он занимается только проектированием. В табл. 1 представлен ряд проектов. Проведено их сравнение по таким параметрам, как язык описания распределенных процессов и способ и тип задания языка.

Табл. 1. **Грид-проекты и языки описания распределенных процессов**

Проект	Язык описания процессов	Тип языка	Объем проекта
FhRG — Fraunhofer Resource Grid [1]	GADL: GResourceDL, GInterfaceDL, GDataDL, GJobDL	Сети Петри	Шесть институтов общества Фраунгофер
Teuta [3, 4]		Поток работ [7]	Один центр разработки
Triana [5]	OGSA [6]	Поток работ	Один центр разработки

Как можно видеть, исследователи в разных странах [7] сосредоточены на предоставлении конечному пользователю инструментов для визуального описания.

В данной работе описывается подход к автоматизации разработки ГРИД-приложений, который используется в разрабатываемом в лаборатории инструментарии. Данный подход предполагает автоматизацию построения концептуального описания ГРИД-приложений и правил трансформации языка описания процессов в код целевой платформы.

2. Описание подхода. Разрабатываемый подход предполагает следующий порядок действий:

— из некоторого формализованного описания языка задания распределенных процессов порождается онтология, из которой происходит конструирование концептуальной модели и кода для работы с ней, для работы с диаграммами описания процессов, создаваемых пользователем;

— с помощью правил трансформации, подаваемых на вход также в формализованном виде, осуществляется порождение кода для трансформации диаграмм пользователя на целевую платформу.

На рисунке приведена схема разрабатываемого подхода.

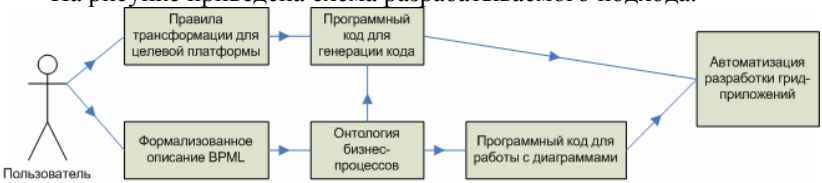

Схема подхода к автоматизации разработки ГРИД-приложений.

На этапе проектирования и разработки инструментария разработчик задает формализованное описание языка моделирования распределенных процессов. В нашем случае в качестве такого языка выступает язык описания бизнес-процессов (BPML), его формализованное описание может быть получено путем анализа спецификации на BPML. Затем введенное разработчиком представление языка описания процессов с помощью набора правил трансформируется в онтологию бизнес-процессов (см. следующий раздел). По построенной концептуальной модели генерируется программный код для работы с диаграммами BPML.

Кроме того, разработчиком задаются правила генерации кода для целевой платформы. В случае использования ГРИД как целевой платформы это будут правила создания программного кода для сервис-

ориентированных приложений. Данные правила применяются к онтологии описания бизнес-процессов для создания программного кода создания приложения по диаграмме пользователя.

Реализация подхода выполнена с использованием средств, предоставляемых платформой Java EE. Кроме того, используются следующие программные средства:

а) Eclipse.UML2Tools — для визуального проектирования диаграмм пользователем и анализа диаграмм с целью построения онтологии,

б) Jena — для операций над онтологиями и выводов на ограничениях онтологий с целью порождения программного кода для целевой платформы.

3. Онтологическая модель. $G = \{g\}$ — множество ГРИД-сервисов, используемых приложением. $C = \{c\}$ — множество вызовов ГРИД-сервисов (далее — сервисов) из приложения. Введем следующие отношения на множестве вызовов:

— $c_i Pc_j, - c_i, c_j - C$ — вызовы двух сервисов выполняются параллельно. Отношение обладает следующими свойствами: нерефлексивно, симметрично и транзитивно.

— $c_i Sc_j, - c_i, c_j - C$ — вызовы двух сервисов выполняются строго последовательно. Отношение нерефлексивно, несемметрично и нетранзитивно.

Введем понятие модуля. Под модулем будем понимать:

а) вызов сервиса;

б) набор сервисов, либо модулей, выполняющихся параллельно;

в) набор сервисов, либо модулей, выполняющихся последовательно или параллельно и выполняющихся по срабатыванию условного оператора;

г) условный оператор и следующий за ним модуль, $M = \{m\}$ — множество модулей.

Введем следующие отношения на множестве модулей:

— $m_i Pm_j, - m_i, m_j - M$ — два модуля выполняются параллельно. Отношение обладает следующими свойствами: нерефлексивно, симметрично и транзитивно.

— $m_i Sm_j, - m_i, m_j - M$ — два модуля выполняются строго последовательно. Отношение нерефлексивно, несимметрично и нетранзитивно.

В дальнейшем будем оперировать, по-возможности, понятием модуль, и не будем говорить о вызове сервисов.

Будем рассматривать операторы условия как отдельное множество $T = \{t\}$. Введем следующие отношения:

— $S - M \times T$ — вызов условного оператора следует строго после модуля. Отношение нерефлексивно, несемметрично и нетранзитивно.

— $St - M \times T$ — модуль выполняется в случае успешного выполнения условного оператора.

Под термином «*онтология*» понимается детальная формализация некоторой области знаний с помощью концептуальной схемы. Она описывается множеством классов, множеством атрибутов классов, множеством доменов атрибутов, множеством отношений и ~~множество~~ множеством правил вывода на отношениях.

Множество отношений включает в себя:

1) собственно отношения (класс, атрибут, домен);

2) таксономические (быть экземпляром) и иерархические (быть частью) отношения между классами;

3) отношения совместимости классов;

4) ассоциативные отношения между классами;

5) ограничения на число классов в подмножестве классов;

6) функциональные ограничения.

Под множеством правила вывода на ограничениях будем понимать набор правил трансформации для целевой платформы, которые описывают правила порождения кода программы.

Объявим следующие классы:

— *модуль* — базовая единица в процессе выполнения;

— *вызов сервиса* наследуется от класса «модуль»;

— *расщепление потока управления* наследуется от класса «модуль», является аналогом элемента fork в языке описания бизнес-процессов;

— *слияние потока управления* наследуется от класса «модуль», является аналогом элемента fork в языке описания бизнес-процессов;

— *условие* наследуется от класса «модуль», является аналогом элемента inclusive decision в языке описания бизнес-процессов;

— *составной модуль* наследуется от класса «модуль», представляет собой композицию других модулей;

— *параллельный модуль* наследуется от «составного модуля», представляет собой группу параллельно выполняющихся модулей;

— *последовательный модуль* наследуется о «составного модуля», представляет собой группу последовательно выполняющихся модулей.

Перечисленные классы обладают следующими четырьмя атрибутами:

1) текущий модуль у класса «модуль» указывает на следующий модуль, подлежащий выполнению после текущего модуля;
2) параметры вызова сервиса у класса «сервис»;
3) успешный модуль у класса «условие» вызывается в случае успешного выполнения условия;
4) родительский модуль у класса «модуль» указывает на наличие модуля, частью которого является данный модуль (например, когда сервис выполняется параллельно с другими сервисами, для него вышестоящим модулем будет параллельный модуль).

Перечисленные атрибуты относятся к следующим доменам:

— для атрибута (1) — все экземпляры класса «модуль» и его наследников,

— для атрибута (2) — URI,

— для атрибута (3) — все экземпляры класса «модуль» и его наследников,

— для атрибута (4) — все экземпляры класса «модуль» и его наследников.

Мы не вводим отдельно ограничения (отношения), а выражаем их через атрибуты.

4. Трансформация онтологии в программный код. Введем обозначения:

$C = \{c\}$ — множество классов в онтологии,

$A_{c_i} = \{a\},\ c_i \in C$ — множество атрибутов классов,

$D_{a_j} = \{d\},\ a_j \in A$ — множество доменов атрибутов класса,

$I = C \times C$ — иерархические отношения между классами,

$O = C \times C$ — прочие отношения между классами.

Опишем трансформации из терминов онтологий в термины реляционной базы данных:

— класс c_i преобразуется в сущность (таблицу) t_j, если он не является подклассом другого класса, т. е. не имеет иерархических отношений с другим классом;

— если класс c_i является подклассом другого класса c_j, то:

а) создается сущность с именем type_of_cj с единственным полем description;

б) в таблицу базового класса добавляется поле id_type_of;

в) заносятся две записи в таблицу type_of_cj: 1) маркер базового класса, 1) маркер текущего класса;

г) создается таблица attr_of_ci. Если уже был встречен подкласс c_j, то выполняются только данный пункт и запись 1 из предыдущего пункта (в).

— атрибут a_j преобразуется в поле таблицы, соответствующей классу, к которому относится атрибут, в случае если класс не имеет базового класса. Если же класс, к которому относится атрибут, является чьим-то подклассом, то поле добавляется к таблице attr_of_ci;

— отношения $c_i O c_j$ реализуются с помощью механизма внешних ключей, причем внешний ключ добавляется в таблицу, соответствующую классу c_i, в случае, когда класс c_i не является чьим-либо подклассом, и в таблицу attr_of_ci — в противном случае.

Схема базы данных для приведенной выше онтологии приведена на листинге 1.

```
CREATE TABLE block (
        Id integer UNIQUE,
        Next integer REFERENCES block (id),
        Id_type_of integer REFERENCES type_of_block (id),
        Parent integer REFERENCES type_of_block (id)
);
CREATE TABLE type_of_block (
        Id ineteger UNIQUE,
        Description TEXT
);
CREATE TABLE attr_of_service (
        Id integer UNIQUE,
        Id_block integer REFERENCES block (id),
        Uri text
);
```

```
CREATE TABLE attr_of_condition (
        Id integer UNIQUE,
        Id_block integer REFERENCES block (id),
        Condition text,
        Suc_next integer REFERENCES block (id)
);
CREATE TABLE composite_block (
        Id integer UNIQUE,
        Id_block integer REFERENCES block (id)
);
CREATE TABLE parallel_block (
        Id integer UNIQUE,
        Id_composite_block integer REFERENCES composite_block (id),
        Id_block integer REFERENCES block (id)
);
CREATE TABLE perial_block (
        Id integer UNIQUE,
        Id_composite_block integer REFERENCES composite_block (id),
        Id_block integer REFERENCES block (id)
);
```

Листинг 1. Пример схемы базы данных для онтологии.

Сходным образом сгенерируем код для разбора диаграммы, поступающей от пользователя.

5.Заключение. В статье описан подход, который позволяет существенно упростить разработку систем автоматизации распределенных приложений. Упрощение достигается за счет частичной автоматизации построения онтологии предметной области и разделения онтологии на две части: онтологии языка описания распределенных процессов и правил порождения кода для целевой платформы.

Литература

1. *Hoheisel A.* Grid Application Definition Language — GADL 0.2 // Technischer Rep., Fraunhofer FIRST, 2002.
2. PNML Refernce Site // [Электронный ресурс] < http://www.pnml.org/> (по состоянию на 04.12.2009)
3. Austrian Grid // [Электронный ресурс] <http://www.austriangrid.at> (по состоянию на 04.12.2009)
4. *Бабошин А.А.* Автоматизация разработки распределенных приложений // Материалы XI Санкт-Петербургской междунар. конф. «Региональная информатика-2008 (РИ-2008)». Санкт-Петербург, 22–24 октября 2008 г. СПб. 2008. С. 300–335.
5. The Triana Project // [Электронный ресурс] <http://www.trianacode.org/> (по состоянию на 04.12.2009)

6. Globus: OGSA — The Open Grid Services Architecture // [Электронный ресурс] <http://www.globus.org/ogsa/> (по состоянию на 04.12.2009)
7. *Бабошин А.А.* Автоматизация разработки распределенных приложений // Информационно-измерительные и управляющие системы. 2008. №10.

Бабошин Андрей Александрович — аспирант ВС ПЗИ. Область научных интересов: ГРИД, онтологии, распределенные системы. Число научных публикаций — 3. andrey.baboshin@gmail.com; 14-я линия В.О., д.39, Санкт-Петербург, 199178, Россия; р.т. +7(812)328-43-69. Научный руководитель — В.И. Воробьев.

Baboshin Andrey — Ph.D. Student, SPIIRAS. Scientific interests: grid, ontology, distributed systems. The number of scientific publications — 3. andrey.baboshin@gmail.com; 14th Line V.O., 39, St.Petersburg, 199178, Russia; office phone +7(812)328-43-69. Supervisor —V.I. Vorobev.

РЕФЕРАТ

Бабошин А. А. **Онтологическое моделирование ГРИД-приложений.**

Современные средства ГРИД-вычислений включают в себя также средства для поддержки автоматизации разработки ГРИД-приложений. Однако, доступные на данный момент средства недостаточно развиты и специфичны для конкретных ГРИД-средств. Цель статьи — продемонстрировать подход разработки средства автоматизации ГРИД-приложений.

В основе подхода лежит идея порождения онтологий из формализованного описания с последующим автоматическим созданием средства автоматизации разработки ГРИД-приложений. Входными данными для такого подхода являются: а) формализованное представление языка описания процессов, с помощью которого пользователь задает ГРИД-приложение; б) правила трансформации описания ГРИД-приложения в код для целевой платформы. Под термином «*онтология*» понимается детальная формализация некоторой области знаний с помощью концептуальной схемы. Она описывается множествами классов, атрибутов классов, доменов атрибутов, отношений и правил вывода на отношениях. Приводится описание конструируемых онтологий с помощью теоретико-множественного аппарата.

Пилотная реализация приведенного подхода выполнена на языке Java и включает в себя использование следующих программных средств: а) Eclipse.UML2Tools — средство создания диаграмм пользователей; б) Jena — библиотека для работы с онтологиями.

SUMMARY

Baboshin A.A. **Ontology modeling of grid-applications.**

Modern tools of grid-calculations includes tools for supporting of automation of grid-application's development. However, currently available tools are not sufficiently developed and are specific to a specific grid resources. Purpose of article is to demonstrate the approach of a project for automation of Grid applications.

In the approach is the idea of ontology's generation from a formalized description, followed by automatic creation of tools for automating the development of Grid applications. The input data for such an approach are: a) a formalized description of the language describing the processes by which the user specifies the grid application; b) the transformation rules describe Grid applications in the code for the target platform. The term «*ontology*» refers to a detailed formalization of some domain knowledge through the conceptual scheme. It is described with the set of classes, the set of classes attributes, the set of attributes domains, the set of relationships, and the set of rules of inference on the relationships. The paper describes the constructed ontology with the set-theoretic apparatus.

Prototype of the above approach implemented in Java and it includes usage of followed program tools: a) Eclipse.UML2Tools is a user-friendly tool for building a diagrams; b) Jena is an framework for manipulating of an ontology.