УДК 004.434;004.453

К. В. Князьков, А. В. Ларченко

# ПРЕДМЕТНО-ОРИЕНТИРОВАННЫЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРИЛОЖЕНИЙ В РАСПРЕДЕЛЕННЫХ СРЕДАХ

Предлагается подход к построению и исполнению композитных приложений в распределенных вычислительных средах на основе предметно-ориентированных описаний. Для его реализации разработаны предметно-ориентированные языки EasyPackage и EasyFlow. На их основе предложены технологии создания предметно-ориентированных интерфейсов, конструирования и исполнения композитных приложений, реализованные в платформе CLAVIRE.

***Ключевые слова:*** *поток заданий, прикладной пакет, композитное приложение, предметно-ориентированный язык, предметно-ориентированный интерфейс.*

**Введение.** Непрерывно возрастающая сложность современных систем распределенных вычислений ограничивает возможности их широкого использования специалистами предметных областей. Как следствие, необходимо развивать специализированные подходы к разработке распределенных приложений с максимальным уровнем абстрагирования от специфики их технической реализации и исполнения на вычислительных ресурсах.

К настоящему времени хорошо зарекомендовал себя подход, основанный на описании сложных вычислительных процессов в форме потоков заданий (workflow, WF). Под WF понимается ориентированный граф, узлами которого, как правило, являются запуски отдельных задач, а ребрами — зависимости между ними, соответствующие операциям обмена данными и управляющими параметрами в распределенной вычислительной среде. В отличие от алго-

ритма, WF не определяет последовательность выполнения операций; он регламентирует только их состав и зависимости.

Существует ряд систем распределенных вычислений, построенных на технологиях работы с WF (их подробную классификацию можно найти в работе [1]). Некоторые системы используют графическую нотацию WF (например, YAWL [2] и Taverna [3]), другие — текстовую в виде скриптов или xml-разметки (например, Condor DAGMan [4], JSDL [5], ADL [6]). Однако возможности эффективного использования и расширения систем в целом ограничены, поскольку в настоящее время не существует унифицированных способов их описания, интерпретации и исполнения в терминах соответствующей предметной области [7].

В настоящей статье предлагается подход к построению и реализации композитных приложений в распределенных вычислительных средах на основе предметно-ориентированных описаний WF. Подход основывается на концепции iPSE [8], что подразумевает необходимость обеспечить максимальное абстрагирование пользователя от технических аспектов исполнения приложения. Пользователь должен описать в форме WF только содержательную составляющую своего приложения; при этом выбор условий выполнения (вычислительных ресурсов, необходимого количества процессов и пр.) осуществляется автоматически [9]. Потому принципиальной задачей является разработка способов человеко-компьютерного взаимодействия, удобных для пользователя и однозначно воспринимаемых системой. Ниже описывается ее реализация посредством двух взаимосвязанных языков: описаний прикладных пакетов (EasyPackage) и описаний структуры WF (EasyFlow).

**Язык описаний прикладных пакетов в распределенной среде.** Простейшая форма WF представляет собой описание исполнения одного вычислительного пакета в распределенной среде с загрузкой входных данных и получением выходных. Однако унифицированное описание этого действия осложнено тем, что разные вычислительные пакеты используют свою стратегию работы с данными (использование конфигурационного файла, командной строки аргументов, переменных окружения, проектов, хранящихся в структуре директорий и файлов). Ситуация осложняется требованием единообразных принципов работы с одним и тем же пакетом, установленным на ресурсах с различными операционными системами, средами управления и исполнения и пр.

В настоящей работе предлагается предметно-ориентированный язык (Domain Specific Language, DSL), позволяющий описывать пакеты в наглядной форме, понятной специалистам-предметникам, и поддающийся программной обработке. EasyPackage разработан на основе реализации языка Ruby (IronRuby) и является интерпретируемым со строгой динамической типизацией и явным приведением типов. Его базовые элементы идентичны элементам Ruby.

Описание пакета представляет собой один или несколько текстовых файлов. Оно использует следующие понятия: пакет, входной/выходной параметр, входной/выходной файл, режим запуска. *Пакет* — это исполняемое приложение, запускаемое в пакетном режиме (модель IPO — Input—Process—Output), которое принимает на входе определенный набор файлов, параметров командной строки, переменных окружения и других источников данных, а на выходе генерирует набор выходных файлов. *Параметр пакета* — это элемент данных, имеющий имя, тип и значение. Параметр может быть входным или выходным, а также может быть параметром исполнения. Тип параметра может быть одним из базовых: строка, логический тип, число с плавающей точкой, перечислимый тип, целое число, список. Режим запуска характеризуется набором используемых в нем параметров.

Структура описания пакета состоит из раздела объявления расширений, общего описания пакета, секционного описания входных и выходных данных пакета (секции *inputs* и *outputs*), описания параметров исполнения (рис. 1). Раздел объявления расширений предназначен для определения процедур, позволяющих расширить функциональные возможности

базовой библиотеки языка. Общее описание пакета включает в себя набор полей, несущих общую информацию о пакете: имя, версия, лицензия, поставщик и т.д. (строки 1—6). Раздел секционного описания содержит определение входных и выходных параметров и файлов. Параметры характеризуются следующим набором полей: тип, значение по умолчанию, процедура проверки значения параметра на корректность (например, параметр в строках 15—21). Параметры могут быть вычислимыми (строки 22—27), тогда для них указывается процедура вычисления из рабочего контекста — *evaluator* (строка 26).

```
1  name "TESTP"
2  display_as "Testp"
3  vendor "SPbSU ITMO"
4  url "http://escience.ifmo.ru"
5  license "GPLv3"
6  description "Simple package example"
7  inputs {
8      raw file {
9          name "inf"
10         filename "arg.txt"
11         place "/"
12         extractor IntegerFileExtractor.new("in")
13         assembler ObjectToSAssembler.new("in")
14     }
15     meta param {
16         name "in"
17         required
18         type int
19         validator lambda { |val, ctx| val > 0 and val < 10000 }
20         validation_error_msg "num have to be in [0; 10000]"
21     }
22     meta param {
23         name "abs_plus_3"
24         required
25         type int
26         evaluator { |ctx| ctx.in.abs + 3 }
27     }
28     cmdline lambda { |ctx| "{0} arg.txt out.txt" }
29 }
30 outputs {
31     auto file {
32         name "output_num"
33         required
34         filename "out.txt"
35         place "/"
36         extractor IntegerFileExtractor.new("out")
37     }
38     auto param {
39         name "out"
40         required
41         display_as "Output number"
42         type int
43     }
44 }
45 prepare_package
```

*Рис. 1*

Контекст работы представляет собой набор уже вычисленных значений параметров (*ctx*). Файловые параметры дополнительно имеют следующий набор полей (строки 8—14): имя файла, путь до файла, процедура извлечения данных из файла (*extractor*), процедура сборки файла (*assembler*). Процедура сборки файла позволяет создавать входной файл, основываясь на значениях входных параметров. На практике используются стандартные процедуры, например, сборка файла по шаблону (библиотека ERB). Процедура извлечения данных из файла, как правило, применяется для выходных файлов пакета с целью определения значений выходных параметров (строка 12). Базовый набор процедур извлечения значений из файлов и их сборки из пара-

метров можно дополнять за счет написания своих процедур в секции расширений. Последним в файле описания является раздел параметров исполнения, который позволяет при работе с пакетом не учитывать неоднородность ресурсов (различных ОС, архитектур). К параметрам данного раздела относятся: скрипт запуска пакета (точнее, процедура его сборки), командная строка, переменные окружения.

Таким образом, описание на языке EasyPackage позволяет не только задать правила обращения к конкретному пакету в распределенной вычислительной среде, но и корректно интерпретировать его входные и выходные данные (посредством процедур *extractor* и *assembler*). Это обеспечивает совместимость (по данным) пакетов различных разработчиков в составе WF.

**Язык описаний WF в распределенной среде.** Описание композитных приложений, состоящих из нескольких прикладных пакетов, требует определения не только правил работы с пакетами, но и структуры взаимодействия между ними. Специализированный язык EasyFlow позволяет упростить процедуру задания композитных приложений. Он предоставляет конечному пользователю гибкие возможности по заданию различных форм WF, в рамках которых выполняются различные прикладные пакеты, происходят генерация выходных данных, их получение, конвертация и обработка.

Характерной чертой языка является полное абстрагирование от особенностей распределенной вычислительной среды, в которой работает пользователь. Фактически EasyFlow — это высокоуровневый язык описания абстрактных workflow (AWF). Такой подход позволяет описывать саму решаемую задачу, а не способ ее исполнения на конкретной вычислительной архитектуре. На рис. 2 приведен пример описания простого AWF, представляющего собой скрипт. Тело скрипта состоит из описания вызовов прикладных пакетов — *шагов*, которые задаются с помощью директивы *step* и представляют собой узлы графа WF. Для описания каждого шага необходимо задать его имя (в примере это *Step1*, *Step2*, *Step3*), название запускаемого пакета (*EmptyPackage*, *Package1* и *Package2*) и перечень предметных параметров этого пакета, который описан в базе пакетов, рассмотренной выше.

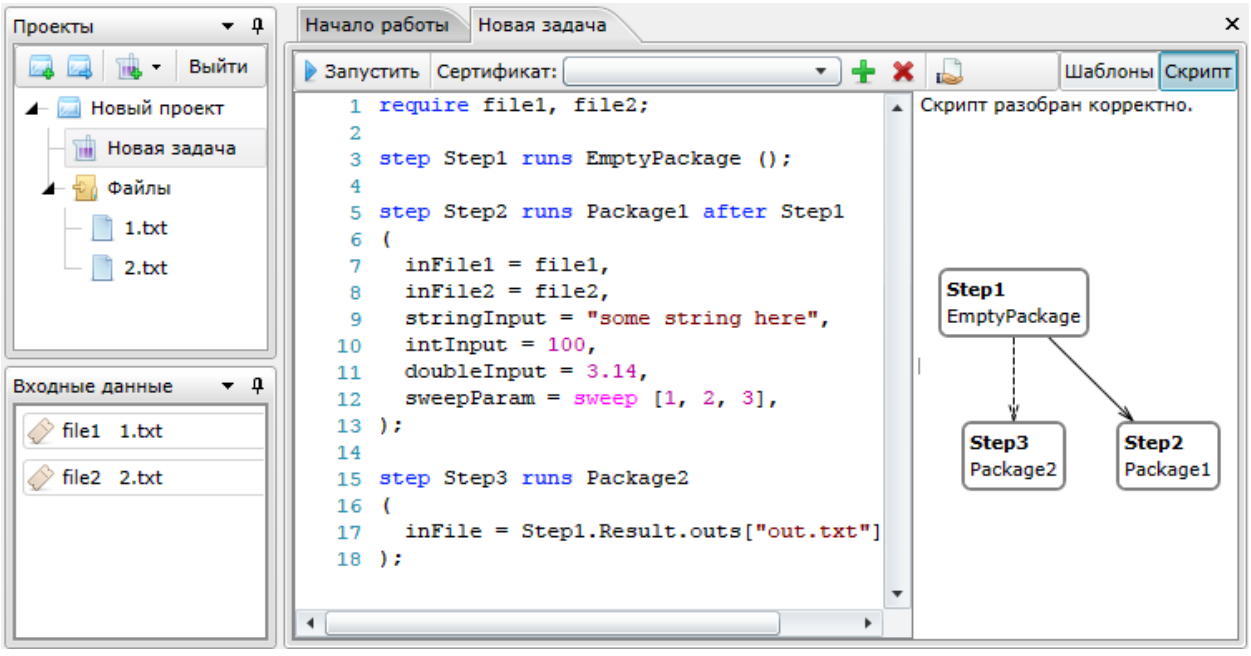


*Рис. 2*

Язык EasyFlow позволяет задавать параметры для следующих типов данных: целое число, строка, число с плавающей точкой, список, структура, указание на использование файла (см. описание шага *Step3*).

Большинство прикладных пакетов, помимо параметров, принимает и генерирует входные и выходные файлы, поэтому в EasyFlow предусмотрена поддержка работы с файлами. Их задание в скрипте представляет собой лишь абстрактное указание с помощью директивы *require*, что освобождает пользователя от необходимости указания абсолютных путей к файлам. В этой директиве через запятую перечислены файловые переменные, которые могут быть указаны в качестве значений параметров при описании шага (см. параметры *inFile1* и *inFile2* в описании шага *Step2*). В рамках одного скрипта директива требования файлов может появляться неограниченное число раз.

Так как WF представляет собой ориентированный граф, в EasyFlow введены механизмы определения порядка выполнения шагов, позволяющие организовать его структуру: зависимости по управлению и зависимости по данным.

*Зависимости по управлению* представляют собой явные указания на то, что один шаг должен начать свое исполнение после завершения другого. Это делается с помощью директивы *after* (см. рис. 2, шаг *Step2*).

*Зависимости по данным* представляют собой неявные указания на зависимости между шагами, которые анализируются при интерпретации скрипта EasyFlow. Они выражаются в том, что некоторые шаги могут использовать данные других шагов, что неявно влияет на последовательность их запуска. Такие зависимости могут присутствовать в описываемом WF одновременно с зависимостями по управлению, что позволяет очень гибко настраивать порядок выполнения шагов. Пример зависимостей по данным содержится в описании шага *Step3* (строка 17), где указано, что в качестве входного файла используется файл „out.txt“, полученный в результате выполнения шага *Step1*.

Еще одной полезной возможностью EasyFlow является автоматическое варьирование параметров (*parameter sweep*). Такая задача часто возникает, когда необходимо запустить один и тот же вычислительный пакет, закрепив одни и варьируя другие параметры. Для этого в язык введена директива *sweep*, которая принимает список параметров для варьирования и из одного шага создает $N$ шагов, где $N$ соответствует числу элементов в декартовом произведении списков варьирования для различных параметров.

Пример варьирования параметра приведен на рис. 2 (строка 12). В этом примере будут запущены три шага *Step2* с параметром *sweepParam*, равным соответственно единице, двум и трем при прочих зафиксированных параметрах.

Таким образом, WF, описанные на языке EasyFlow, полностью независимы от конкретной архитектуры вычислений и хранения данных, что позволяет пользователям распределенной среды беспрепятственно обмениваться ими и запускать их на различных вычислительных ресурсах.

**Предметно-ориентированные интерфейсы к распределенным приложениям.** При использовании представленных выше языков для описания пакетов и WF возможно ввести еще один уровень абстракции в представлении вычислительных задач для пользователей — предметно-ориентированные интерфейсы (ПОИ). Такой подход основан на разделении этапов конструирования WF и его использования, когда экспертом предметной области заранее задаются некие шаблоны применения пакетов, которые затем автоматически трансформируются в простые динамически генерируемые графические интерфейсы, доступные пользователям посредством сети Интернет. С их помощью пользователь может задать необходимые параметры расчета, запустить задачу на исполнение и получить результаты. Примером использования такого подхода может являться выполнение множества однотипных инженерных расчетов, когда от запуска к запуску требуется варьировать только ограниченный набор параметров.

При автоматическом построении ПОИ могут быть целиком использованы формальные описания пакетов на языке EasyPackage: набор параметров, их возможные значения и ограни-

чения на них, взаимосвязь между параметрами и пр., что позволяет реализовать следующие возможности: отображение альтернатив при задании параметров, подсказки при выборе значений параметров, проверка значений и полноты состава параметров, отображение результатов вычислений. Таким образом, проблемно-ориентированный интерфейс приобретает все необходимые функциональные возможности, свойственные графическим интерфейсам самих прикладных пакетов.

Принципиальной особенностью технологии ПОИ, основанной на использовании языков EasyFlow и EasyPackage, является автоматизация процесса построения и поддержки интерфейсов, применимая для пакетов из различных предметных областей. Как следствие, такой подход наиболее предпочтителен для коллаборативных и глобальных распределенных систем (например, Грид-сред), в которых появление новых ресурсов и пакетов определяется интересами самих пользователей. При этом обеспечивается унификация не только по описаниям пакетов, но и по визуальным формам интерфейсов, что принципиально для быстрого освоения пользователями новых пакетов.

**Исполнение композитных приложений в распределенной среде.** Рассмотренные выше подходы к описанию пакетов, композитных приложений и интерфейсов работы с ними реализуются многофункциональной инструментально-технологической платформой CLAVIRE [10]. Композитное приложение в рамках платформы CLAVIRE представляется в виде скрипта описания WF на языке EasyFlow, который может быть параметризован набором входных параметров и файлов, а также параметров исполнения, т.е. один и тот же WF может быть исполнен для разного набора входных данных, а также в различных условиях исполнения. За разбор скрипта WF и за исполнение WF в целом отвечает компонент интерпретации. После получения скрипта данный компонент „разбирает“ его и преобразует во внутреннее представление (предварительно проверив его корректность). Обработка представления WF производится непрерывно, согласно событийной модели функционирования, т.е. интерпретация WF происходит в рамках цикла обработки поступающих событий. При запуске отдельной задачи происходит интерпретация параметров узла WF и формируется описание запуска задачи, после чего сформированное описание передается в очередь компонента исполнения WF. Для определения параметров исполнения данной задачи привлекается компонент планирования, который для определения очередности исполнения задач использует данные моделей производительности пакета и данные о текущей загруженности вычислительных ресурсов. Далее компонент исполнения подготавливает данные для пакета и производит запуск пакета на конкретном ресурсе, после чего обрабатывает выходные данные. Для подготовки пакета к запуску и обработки его результатов используется база пакетов, которая позволяет сопоставить абстрактные и фактические правила работы с каждым пакетом в составе WF. На рис. 3, *а* приведена структура компонента — базы пакетов. Он состоит из интерфейсной библиотеки и репозитория пакетов. Описание пакета в такой схеме хранится в виде файла со скриптом EasyPackage в репозитории. При этом компоненты системы для работы с данным описанием получают скрипты и сопутствующие файлы и интерпретируют их на своей стороне, используя лишь необходимую им информацию. Данный подход выгодно отличается от применения централизованного хранилища информации о пакетах, построенного на сервисно-ориентированной модели, так как позволяет легко масштабировать систему на большее количество пользователей за счет перераспределения нагрузки. На рис. 3, *б* представлен сценарий использования базы пакетов при запуске задачи. Необходимо отметить, что характеристиками на данной схеме считаются вычислимые параметры, необходимые только для планирования запуска в распределенной среде. После получения и обработки результатов данные о завершении работы WF передаются обратно в интерпретатор,

там они становятся доступными пользователю (через соответствующий интерфейс — графическую оболочку CLAVIRE или ПОИ).

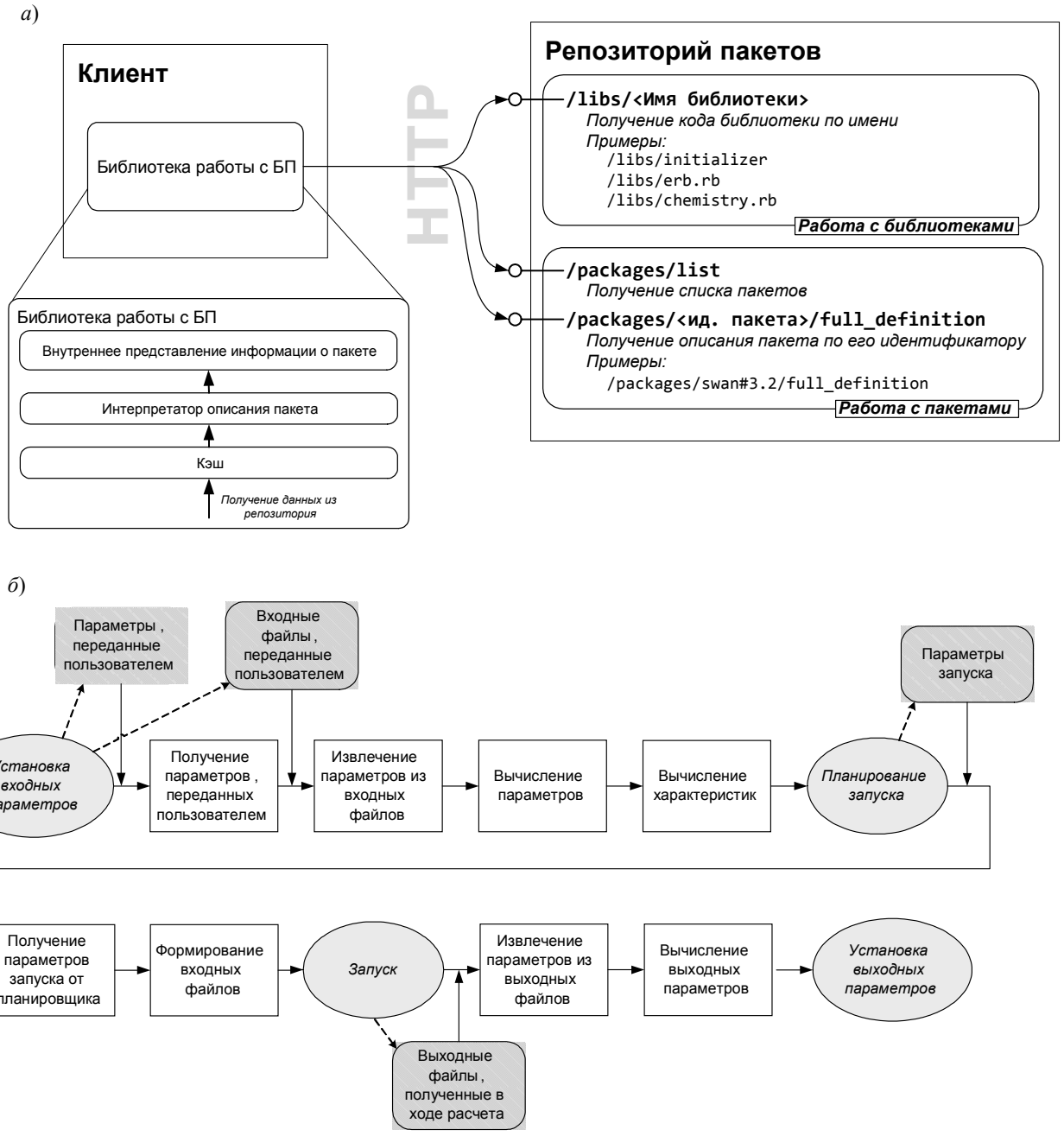


*Рис. 3*

**Заключение.** В настоящей работе предложены подходы к описанию предметно-ориентированных пакетов и композитных приложений в форме WF. Они обеспечивают эргономичность, гибкость и расширяемость распределенных сред для компьютерного моделирования и обработки данных в рамках парадигмы eScience. Данные подходы реализованы и апробированы в составе многофункциональной инструментально-технологической платформы CLAVIRE.

Работа выполнена в рамках проектов по реализации Постановлений № 218 и 220 Правительства Российской Федерации при поддержке ФЦП „Научные и научно-педагогические кадры инновационной России на 2009—2013 гг.“.

## СПИСОК ЛИТЕРАТУРЫ

1. *Yu J., Buyya R.* A Taxonomy of Workflow Management Systems for Grid Computing // J. of Grid Computing. 2005. Vol. 3, N. 3—4. P. 171—200.
2. *Van der Aalst W.M.P., ter Hofstede A.H.M.* YAWL: Yet Another Workflow Language (Revised version) [Электронный ресурс]: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.104.3819&rep=rep1&type=pdf>.
3. Taverna — open source and domain independent Workflow Management System [Электронный ресурс]: <http://www.taverna.org.uk/>.
4. DAGgMan Applications [Электронный ресурс]: <http://www.cs.wisc.edu/condor/manual/v7.0/2_10DAGMan_Applications.html>.
5. Job Submission Description Language (JSDL) Specification, Version 1.0 [Электронный ресурс]: <http://www.gridforum.org/documents/GFD.56.pdf>.
6. *Guidolin M., Brady T., and Lastovetsky A.* How Algorithm Definition Language (ADL) Improves the Performance of SmartGridSolve Applications // The 7th High-Performance Grid Computing Workshop. Atlanta, USA, 2010.
7. *Goderis A., Sattler U., Lord P., Goble C.* Seven Bottlenecks to Workflow Reuse and Repurposing // The Semantic Web – ISWC 2005- 2005. P. 323—337.
8. *Бухановский А. В., Ковальчук С. В., Марьин С. В.* Интеллектуальные программные комплексы компьютерного моделирования сложных систем: концепция, архитектура и примеры реализации // Изв. вузов. Приборостроение. 2009. Т. 52, № 3. С. 5—24.
9. *Бухановский А. В., Васильев В. Н.* Современные программные комплексы компьютерного моделирования e-Science // Там же. 2010. Т. 53, № 3. С. 60—64.
10. *Бухановский А. В., Васильев В. Н., Виноградов В. Н., Смирнов Д. Ю., Сухоруков С. А., Яппаров Т. Г.* CLAVIRE: перспективная технология облачных вычислений второго поколения // Там же. 2011. Т. 54, № 10. С. 7—14.

***Сведения об авторах***

***Константин Валерьевич Князьков*** — НИИ Наукоемких компьютерных технологий Санкт-Петербургского государственного университета информационных технологий, механики и оптики; младший научный сотрудник; E-mail: constantinvk@gmail.com

***Алексей Викторович Ларченко*** — канд. техн. наук; НИИ Наукоемких компьютерных технологий Санкт-Петербургского государственного университета информационных технологий, механики и оптики; старший научный сотрудник; E-mail: aleksey.larchenko@gmail.com

Рекомендована НИИ НКТ

Поступила в редакцию
15.05.11 г.