

ТЕХНОЛОГИЯ МОДЕЛИРОВАНИЯ РАСПРЕДЕЛЕННЫХ СИСТЕМ, ОСНОВАННАЯ НА ВИЗУАЛЬНОМ ЯЗЫКЕ, И ЕЕ ПРИЛОЖЕНИЯ

© 2004 С.В. Востокин

Самарский государственный аэрокосмический университет

В статье рассматривается новая технология моделирования распределенных систем, предназначенная для использования в качестве основы сред и средств автоматизации параллельного программирования. Описывается семантика, набор графических примитивов и текстовая нотация визуального языка моделирования. Рассматриваются возможные сферы применения предлагаемого метода моделирования.

Введение

В последнее время в компьютерной индустрии наметилась тенденция к интеграции вычислительных ресурсов для решения сложных задач. Она обусловлена развитием и широким распространением технических средств, таких как высокопроизводительные компьютеры и коммуникационные среды высокой пропускной способности. Сложность и пространственно распределенный характер протекающих в них процессов порождает комплекс технических проблем, включающий проблемы надежности, масштабируемости алгоритмов и программ, мобильности программ, эффективности, повторного использования кода [1]. В связи с этим разработка качественного программного обеспечения для распределенных систем не возможна без применения подхода, опирающегося на формальную модель.

Известные в настоящее время методы моделирования можно разделить на два класса. К первому классу относятся логические методы описания и анализа распределенных систем. Примером является широко используемый аксиоматический метод доказательства корректности параллельных алгоритмов, предложенный в работе [2], и специальные разновидности модальных логик, например, темпоральная логика Лампорта [3]. Недостатком логического подхода применительно к конечной цели моделирования (разработке программного обеспечения распределенных систем) является то, что данный подход не дает ответа на вопрос, как реализуется опи-

сываемое моделью поведение системы.

Модели, основанные на императивном описании вычислительного процесса, лишены указанного недостатка логических моделей. Наиболее известными моделями, принадлежащими к данному классу, являются сети Петри и их разновидности [4] и взаимодействующие последовательные процессы Хоара [5]. Однако использование традиционных императивных моделей в качестве основы средств автоматизации программирования порождает ряд проблем. При описании реальных распределенных процессов обработки и управления сталкиваемся со структурной сложностью модели. Это вызвано тем, что для таких процессов формальная запись становится громоздкой. В традиционных моделях также не рассматривается важный аспект промышленного программирования, связанный с повторным использованием спецификаций. Кроме этого, для автоматической обработки модели с целью проверки ее корректности и синтеза исполняемой спецификации вычислительного процесса должен быть формализован язык описания модели.

В статье предлагается новый метод описания сложных процессов, направленный на преодоление отмеченных выше недостатков традиционных моделей распределенных систем. Метод основан на представлении логики вычислительного процесса в виде иерархической графовой структуры, описываемой на специальном текстовом языке. Подробно метод представлен в работе [6].

Семантика модели

Пусть требуется формально описать некоторую дискретную систему. С течением времени система переходит из одного дискретного состояния в другое. Поведение дискретной системы можно описать как последовательность смены состояний, начиная с некоторого начального состояния:

$$\Sigma = \langle S_0, S_1, S_2, S_3, \dots \rangle.$$

Понятие *состояние* при описании дискретных систем связано с понятием переменной. *Переменная* есть функция, вычисляющая значение состояния по имени: $V(x) : X \rightarrow S$. Состояние системы также удобно считать дискретным, представляющим собой суперпозицию состояний, хранящихся в наборе переменных: $S = \langle v(x_1), v(x_2), v(x_3) \dots v(x_n) \rangle$. В данном методе моделирования применяется конечный набор переменных для представления состояния дискретной системы.

Смена состояния системы происходит при выполнении *действий*. Под действием понимается некоторая функция, сопоставляющая заданному “старому” состоянию “новое” состояние: $A(s) : S \rightarrow S$. Дополнительно действия обладают следующими двумя особенностями. Действия локальны в том смысле, что они влияют лишь на часть состояния системы. Следовательно, каждое отдельное действие связано с подмножеством переменных системы. Действия понимаются как атомарные операции. В нашей модели это означает, что никакие два действия не могут одновременно работать с одной и той же переменной. При моделировании используется конечное множество действий с фиксированным набором переменных для каждого действия.

Обобщенную структуру дискретной системы можно представить в виде рис.1. На

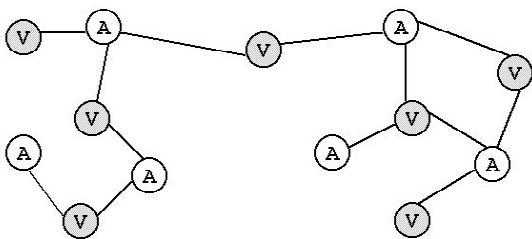


Рис.1. Дискретная система, состоящая из действий и переменных

рисунке показаны переменные и действия некоторой дискретной системы и отношения между ними.

Действия дискретной системы сгруппированы в *процессы*. При этом каждое действие относится к одному и только одному процессу. Не существует двух процессов, одновременно включающих некоторое действие:

$$\forall A, \exists P : A \subset P$$

и

$$\forall P_1, \forall P_2 : P_1 \neq P_2 ! \exists A : A \subset P_1 \wedge A \subset P_2 .$$

Основное свойство процессов заключается в том, что все действия процесса происходят последовательно во времени. Выполнение некоторого действия может начаться только после завершения текущего активного действия. С учетом рассмотренного ограничения модель будет выглядеть, как показано на рис.2.

Группировка действий в процессы неявно определяет внутреннее состояние процесса. Если с некоторой переменной модели v работают действия, принадлежащие одному процессу, то такая переменная является частью состояния процесса. Переменные, с которыми связаны действия, принадлежащие разным процессам, служат для организации взаимодействия процессов. На рис.2 имеется одна такая переменная.

Для того, чтобы модель могла описывать пространственно-распределенные системы, необходимо организовать взаимодействие процессов при помощи сообщений, а не только с помощью разделяемых переменных, как показано на рис.2. Такое взаимодействие организуется очевидным образом, если ввести дополнительное ограничение на порядок

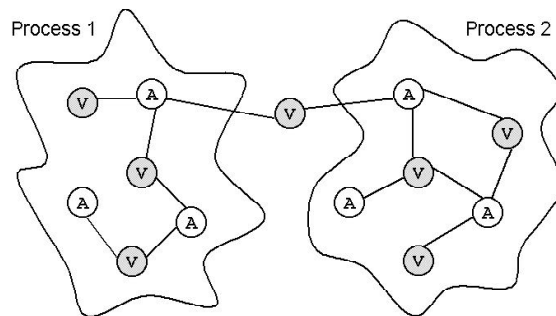


Рис.2. Дискретная система с группировкой действий в процессы

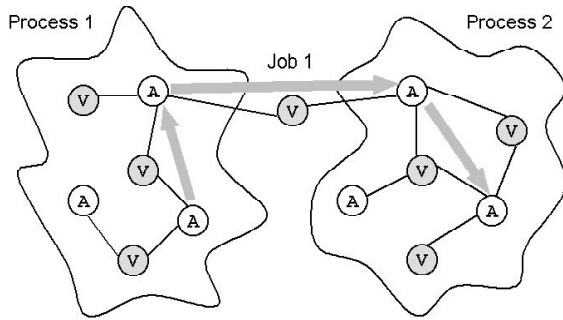


Рис.3. Дискретная система с группировкой действий в процессы и работы

следования действий во времени, которое задается при помощи понятия *работы*.

Под работой понимается такое причинно-следственное упорядочивание действий (возможно принадлежащих разным процессам), при котором любая переменная модели, не являющаяся частью состояния процесса, принадлежит к одной и только одной работе. Такие переменные являются частью состояния работы. Пример упорядочивания действий показан на рис.3.

При наличии линейного упорядочивания действий во времени, задаваемого работами, замена разделяемых переменных на сообщения организуется следующим образом. Вместо разделяемой процессами переменной в каждом из связываемых данной переменной процессов заводится новая переменная. При переходе соответствующей работы из процесса в процесс происходит посылка сообщения, в котором передается значение из одной локальной копии разделяемой переменной в другую, как показано на рис. 4.

Следующий тип ограничений направлен на преодоление структурной сложности, возникающей при описании пространственно

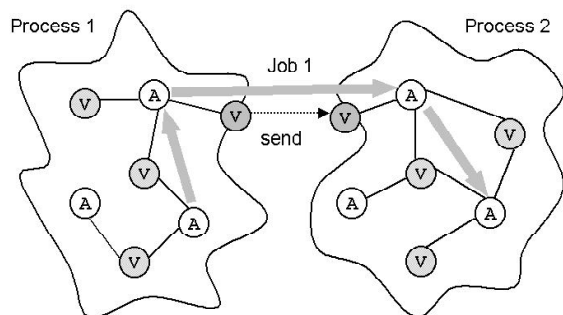


Рис.4. Дискретная система с взаимодействием процессов при помощи сообщений

распределенных процессов. Реальные дискретные системы при данном способе описания состоят из сотен и тысяч процессов. Например, это имеет место при решении задачи на массово-параллельном компьютере. В этом случае полностью описать всю сеть из взаимодействующих процессов в явном виде не представляется возможным. Однако, в структуре процессов можно выделить повторяющиеся фрагменты, для которых возможно обобщенное описание при помощи того или иного метода абстрагирования.

Метод абстрагирования, используемый в нашей модели, требует, чтобы структура взаимодействия процессов была иерархически упорядоченной, как показано на рис.5. Это означает, что все процессы относятся к некоторому уровню иерархии. Каждый из процессов может взаимодействовать только с одним процессом, стоящим на один уровень выше по иерархии. Любой из процессов может взаимодействовать с произвольным числом процессов, стоящих на один уровень ниже по иерархии. Имеется главный процесс, находящийся на самом верхнем уровне иерархии. То есть, структура взаимодействия процессов представляет собой дерево с корнем в главном процессе рис.5.

Как конкретно осуществляется обобщение структуры процессов, а также как задается порядок действий внутри процессов, определяется языком моделирования. Синтаксис языка моделирования построен таким образом, чтобы автоматически выдерживать структурные ограничения, представленные на рис.1–5, и включает механизмы абстракции, позволяющие компактно описывать системы большой структурной сложности.

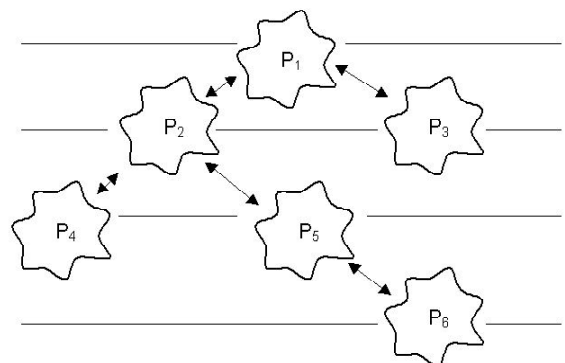


Рис.5. Дискретная система с ограничением на взаимодействие процессов

Синтаксис модели

Для описания синтаксиса языка моделирования применяются Расширенные Бэкуса-Наура Формы (РБНФ). В нотации РБНФ используются следующие символы. Правая часть предложения отделяется от левой части двойным равенством (==). Конец предложения обозначается точкой (.). Варианты разделяются знаком |. Квадратные скобки [и] означают необязательность записанного внутри них выражения. Фигурные скобки { и } означают повторение записанного внутри выражения (возможно 0 раз). Круглые скобки (и) используются для задания порядка связывания символов РБНФ. Если это специально не предписано скобками, обычный порядок связывания — слева направо. Запись "a.."z" означает любой символ в диапазоне от a до z.

Нетерминальные символы РБНФ заключены в угловые скобки < и > (например, <process>). Терминальные символы записываются целиком заглавными буквами (например, PROCESS) или заключаются в кавычки (например, " ").

Ниже представлено полное описание предлагаемого языка моделирования, заданное порождающей грамматикой в нотации РБНФ.

```

<str> ==
    строка символов с \\ и \" внутри;
    продолжение строки: "xxx" "xxx"
    воспринимается как "xxx xxx".
<num> == ("1".."9"{"0".."9"})|"0".
<label> == "@{"a".."z"|"A".."Z"|"
    "0".."9"}.
<ident> == ("a".."z"|"A".."Z")
    {"a".."z"|"A".."Z"|"0".."9"}.
<id_def> == <ident> [EXPORT].
<id_use> == <ident> [FROM <ident>].
<module> == MODULE <ident>
    [PARAM <ident>{,<ident>}]
    [COMMENT <str>]
    {
        <import>|
        <type>|<var>|
        <job>|<proc_ref>|
        <action>|<condition>|
        <port>
    }
    END.
<import> ==
    IMPORT <ident> [AS <ident>]
    [PARAM <ident>{,<ident>}].
<type> ==

```

```

    TYPE <id_def> [COMMENT <str>].
<var> ==
    VAR <id_def> TYPE <id_use>
    [COMMENT <str>].
<job> ==
    JOB <id_def>
    [INCLUDES <id_use> {"","<id_use>}]
    [COMMENT <str>].
<proc_ref> == PROCESS <ident>.
<action> ==
    ACTION <id_def> [BASE <id_use>]
    [COMMENT <str>]
    [PARAM <id_use>[IN|OUT|IO]
    {"","<id_use>[IN|OUT|IO]}]
    [PUT <id_use>[TO <id_use>]
    {"","<id_use>[TO <id_use>]}]
    [GET <id_use>[TO <id_use>]
    {"","<id_use>[TO <id_use>]}]
    END.
<condition> ==
    CONDITION <id_def> [BASE <id_use>]
    [COMMENT <str>]
    [PARAM <id_use>{"","<id_use>}]
    END.
<process> ==
    PROCESS <id_def> ENTRY <label>
    [COMMENT <str>]
    {<proc_com>|<node>}
    END.
<proc_com> ==
    TEXT <str> [PIC <num> "," <num>].
<node> ==
    <label> (CALL <id_use> | STUB)
    [EXIT [<num>]]
    [PIC <num> "," <num>]
    {
        (IF <id_use> THEN | CONTINUE)
        <label> [PIC <num> "," <num>] |
        (NEW <id_use> TO <label>
            [PIC <num> "," <num>])
    }
    }.
<port> ==
    PORT <id_def> [COMMENT <str>]
    [OUTPUT <id_use>{"","<id_use>}]
    [INPUT <id_use>{"","<id_use>}]
    END.

```

Представленный язык моделирования обладает следующими особенностями. Грамматика языка сводима к автоматной грамматике, что делает очевидным процедуры лексического и синтаксического анализа. Язык абстрагируется от формы реализации последовательных процедур обозначаемых *действиями*. Для этой цели используются предложения с нетерминальными символами <type>, <action> и <condition> в левых частях. Для описания взаимодействия с внешней си-

стемой используются порты, представленные нетерминальным символом <port>.

В качестве основного механизма абстрагирования используется концепция модулей. Модули описываются нетерминальным символом <module>. Модули связываются друг с другом, образуя два уровня иерархии. Первый уровень реализует отношение экспорт-импорт. Для его выражения в языке используются терминальные символы EXPORT и IMPORT. Семантика этого отношения соответствует модулям в языках программирования: часть структурных элементов модели, описанных в модуле, объявляется доступным в других модулях. Для того чтобы использовать структурные элементы из одного модуля в другом, модуль должен быть импортирован при помощи предложения <import>. Второй уровень иерархии модулей реализует отношение параметризации: один модуль может быть параметром другого модуля. Формальные параметры модулей определяются после слова PARAM в предложении нетерминального символа <module>. Фактические параметры определяются после слова PARAM в предложении нетерминального символа <import>. Механизм параметризации модулей похож на механизм шаблонов в языках программирования.

Логiku работы процессов описывают предложения с нетерминальными символами <process>, <proc_com> и <node> в левых частях. Особенностью этого описания является то, что оно задает аннотированный граф. Благодаря наличию координатной привязки элементов описания графа, задаваемой после терминального символа PIC, управляющий граф процесса может быть представлен семантически эквивалентной диаграммой.

Графическая нотация

Графическая нотация используется для наглядного изображения логики работы процессов. Диаграммы процессов представляют собой аннотированные ориентированные графы. Вершины графа обозначают действия или связи с процессами нижнего уровня иерархии (рис.5). Дуги на диаграммах обозначают последовательность запуска дей-

ствий. Пометки дуг позволяют связать выбор следующего действия с текущим состоянием процесса или типом активной работы.

В графической нотации используется четыре пиктографических символа вершин и два символа дуг. Взаимное расположение символов на плоскости диаграммы определяется координатной привязкой их центров. Смысловая интерпретация пиктографических символов зависит от типа структурного элемента, помечающего данный символ. Ниже описаны все пиктографические символы, их возможные интерпретации и эквивалентная текстовая запись.

Обычная вершина, в зависимости от типа структурного элемента, указанного в метке, выполняет вызов процесса, действия или порта на исполнение. Если вершина не имеет исходящих дуг, то после возврата из действия, порта или процесса, указанных в метке вершины, работы переходят в процесс верхнего уровня согласно иерархии рис.5. Пример пиктограммы и соответствующего кода вершины показан на рис.6.

Начальная вершина аналогична по свойствам обычной вершине процесса. Дополнительно к свойствам обычной вершины, эта вершина является точкой входа для работ, вызывающих данный процесс (переходящих в данный процесс с верхнего уровня иерархии рис.7).

Вершина завершения работы, в зависимости от типа структурного элемента указанного в метке, выполняет вызов процесса, действия или порта на исполнение. После возврата из действия, порта или процесса работа завершается. Из вершины завершения работ могут исходить только дуги, планирующие новые работы. Планирование новых ра-


Node@4 ; **обычная вершина**
 @ CALL Node PIC 10,20

Рис. 6. Обычная вершина процесса


Node@1 ; **начальная вершина**
 .. ENTRY @1 ..
 @ CALL Node PIC 10,20

Рис. 7. Начальная вершина процесса

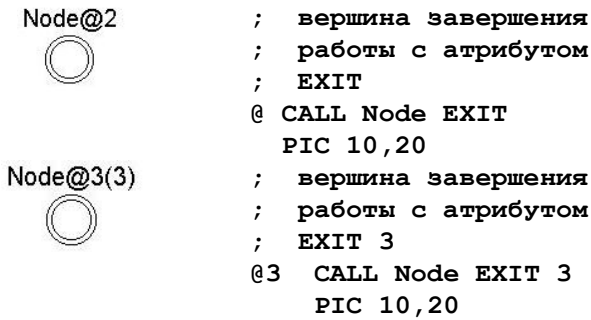


Рис. 8. Вершина завершения работы

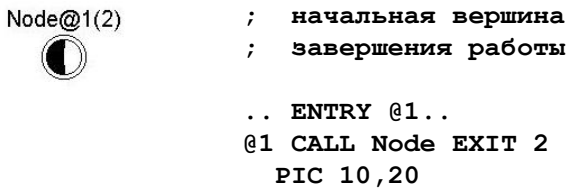


Рис. 9. Начальная вершина, являющаяся вершиной завершения работы

бот выполняется каждый раз по завершении в вершине количества работ, указанных в атрибуте EXIT. Примеры пиктограмм и соответствующих кодов вершин показаны на рис.8.

Начальная вершина, являющаяся вершиной завершения работы, показанная на рис.9, аналогична по свойствам вершине завершения работы. Дополнительно к свойствам вершины завершения работы, эта вершина является точкой входа для работ, вызывающих данный процесс.

Вершины-заглушки не имеют текстовой метки. Вершина-заглушка может использоваться для эскизного проектирования логики процесса, а затем преобразовываться в вершину с меткой. В некоторых случаях вершина требуется для обозначения узловых точек в процессе, концевых вершин, а также для улучшения читаемости диаграммы процесса. Примеры пиктограмм вершин-заглушек показаны на рис.10.

Из вершины может исходить одна или несколько дуг. Дуги имеют следующие атрибуты: текстовая пометка; метка вершины, в

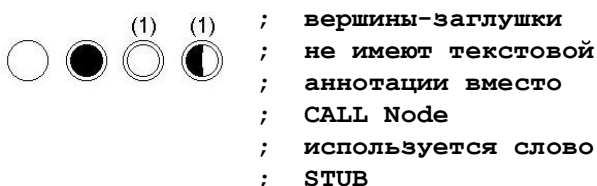


Рис.10. Вершины-заглушки

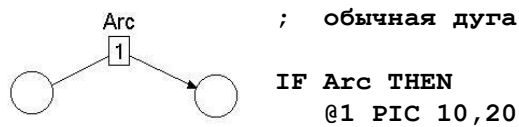


Рис. 11. Обычная дуга процесса

которую ведет данная дуга; координата пиктограммы дуги. Наличие текстовой пометки определяется типом дуги. Пиктограмма дуги располагается в точке ее перегиба. Число на пиктограмме обозначает приоритет дуги. Приоритет дуги в текстовой нотации соответствует порядковому номеру предложения, обозначающего дугу. В зависимости от использования атрибутов различают три типа дуг.

Предложение *обычной дуги* начинается со слова IF, за которым указывается текстовая метка дуги. В качестве метки может использоваться имя условия или работы. При помощи обычных дуг задается порядок выполнения действий процесса. Выбор следующего действия зависит от текущих значений условия или типа работы, помечающей данную дугу рис.11.

В дуге “*тождественно истинный предикат*” не используется текстовая метка (рис.12). Свойства дуги аналогичны свойствам обычной дуги, в которой в качестве метки указано тождественно истинное условие. Дуга может использоваться для эскизного проектирования логики процесса, а затем преобразовываться в дугу с меткой. В готовом процессе дуга используется в качестве последней дуги в группе дуг, исходящих из вершины, для задания перехода по умолчанию.

При исполнении *дуги планирования работы* переход по дуге не выполняется, а создается новая работа с типом, указанным в текстовой метке дуги. Новая работа начнет исполнение с вершины, в которую ведет дуга. Далее вызывающая дугу работа приступает к обработке следующей дуги. Пример пиктограммы и соответствующего кода показан на рис.13.

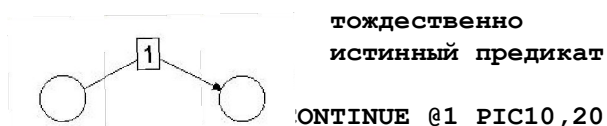


Рис. 12. Дуга — тождественно истинный предикат

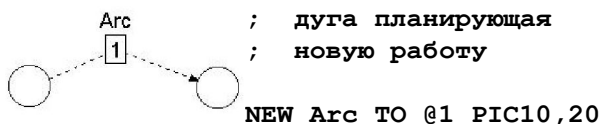


Рис.13. Дуга планирования новой работы

Сферы применения языка моделирования

Помимо академического интереса, заключающегося в разработке новых методов программирования и анализа распределенных вычислительных процессов, мы видим практическое применение данной технологии в следующих областях.

Высокопроизводительные вычисления.

Преимущества от использования языка в данной области заключаются в наглядном представлении и возможности автоматической проверки корректности программ. Механизмы абстрагирования языка позволяют строить типовые алгоритмические структуры (шаблоны) параллельных вычислительных процессов. В настоящее время разрабатывается набор шаблонов для решения задач сеточными методами и библиотека распространенных шаблонов параллельных процессов. Пример диаграммы процесса, относящегося к данной предметной области, показан на рис. 14.

Управление технологическими процессами и сбором данных. Преимущества языка в данной области обусловлены наличием мощных механизмов абстрагирования, таких как модули, наследование и параметризация. Они позволяют описывать сложные системы, состоящие из большого числа компьютеров

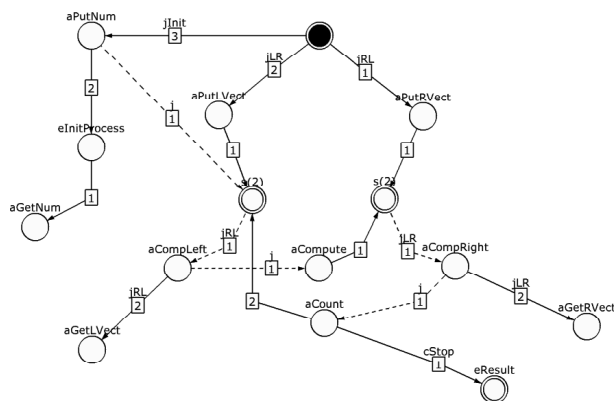


Рис. 14. Пример диаграммы процесса

и контроллеров. Достоинством является унифицированная и простая в освоении нотация языка. Структурные особенности языка позволяют проводить имитационное моделирование и сертификацию программ на соответствие критериям реального времени.

Распределенная обработка в глобальной сети Internet. Одной из главных проблем распределенной обработки в GRID системах является обеспечение надежной работы программы в потенциально ненадежной гетерогенной вычислительной среде. Особенности языка позволяют отделить содержательную часть алгоритмов от сервисных операций резервирования ресурсов, сохранения промежуточных результатов вычислений и трасс вычислений, динамического управления процессами. Благодаря этому GRID-система с точки зрения программиста выглядит как обычный параллельный компьютер.

Управление потоком работ в организациях. Потоки работ — это разновидность параллельных процессов. В языке моделирования имеются сущности, аналогичные сущностям “исполнитель”, “должность”, “работа”, “документ”, поэтому его можно применять в данной предметной области. Дополнительно семантический контроль модели гарантирует безошибочность и непротиворечивость процедур управления. Средства имитационного моделирования позволяют оценить время завершения работ и рассчитать их себестоимость.

Построение пакетов алгоритмов для разных предметных областей. Шаблоны языка позволяют для некоторой предметной области сформировать набор модулей, комбинируя которые (подставляя одни модули в другие) можно строить разные конкретные алгоритмы обработки. В общем случае для использования метода моделирования подходит любая задача, где присутствуют четкие схемы потоков данных, а алгоритмы в узлах обработки можно варьировать. Это, например, задачи цифровой обработки сигналов, задачи проектирования и оптимизации.

Заключение

В статье представлен язык моделирования пространственно - распределенных дис-

кретных систем, достоинствами которого являются компактность, наличие мощных механизмов абстрагирования и наглядность графической нотации. Данная модель является основой для разработки автоматических методов семантического анализа вычислительных процессов и кодогенерации, находящихся в настоящее время в стадии разработки.

СПИСОК ЛИТЕРАТУРЫ

1. *Foster I., Kesselman C., Tuecke S.* The Anatomy of the Grid: Enabling Scalable Virtual Organizations // International J. Supercomputer Applications. 2001. 15(3)
2. *Owicki S., Gries D.* An axiomatic proof technique for parallel programs // Acta Informatica. 1976. 6(4)
3. *Lamport L.* The temporal logic of actions // ACM Transactions on Programming Languages and Systems. 1994. 16(3):
4. *Котов В.Е.* Сети Петри. М.: Наука, 1984.
5. *Hoare C.A.R.* Communicating sequential processes // Communications of the ACM. 1978. 21(8)
6. *Востокин С.В.* Метод описания пространственно распределенных параллельных процессов. Самара: СНЦ РАН 2004.

TECHNOLOGY FOR MODELING DISTRIBUTED SYSTEMS BASED ON VISUAL LANGUAGE AND ITS APPLICATIONS

© 2004 S.V. Vostokin

Samara State Aerospace University

The article presents a new technology for modeling distributed systems. The model is used as a formal basis of integrated development environments and tools for distributed and parallel programming. Semantics, set of graphical primitives, and textual notation of the modeling language are described. Possible implementation areas of the proposed modeling method are discussed.