Языковые средства описания распределенных вычислений в инструментальном комплексе DISCOMP

А. Г. Феоктистов, И. А. Сидоров

В статье рассматриваются языковые средства описания схемы решения задачи в распределенной вычислительной среде модульного программирования, реализованные в инструментальном комплексе DISCOMP (DIStributed COmputing system of Modular Programming). Описываются особенности входного языка и структуры внутреннего представления параллельного вычислительного процесса.

1. Введение

Развитие технологий параллельного программирования привело к существенным изменениям архитектуры пакетов прикладных программ и, как следствие, к появлению новых требований к средствам описания и обработки схем решения задач для конечных пользователей этих пакетов.

Пакеты, включающие в состав своего функционального наполнения параллельные программы, зачастую, разрабатываются для работы в распределенной вычислительной среде (РВС), рассматриваемой нами в традиционном ее понимании (см., например, [1, 2]) как совокупность вычислительных узлов, объединенных коммуникационной сетью. Вычислительный узел представляет собой программно-аппаратный ресурс, включающий модуль оперативной памяти, один или несколько процессоров, жесткий диск и системное программное обеспечение (комплекс программных средств, поддерживающих функционирование узла в РВС). Один из вычислительных узлов назначается главным (управляющим) узлом и наделяется функциями управления заданиями пользователей и ресурсами РВС.

Можно выделить два основных подхода к организации функционального наполнения пакета и решению прикладных задач в РВС.

1) Все прикладные программы пакета находятся в одном узле РВС. По схеме решения задачи производится синтез целевой параллельной программы с последующей ее компиляцией и запуском на распределенной вычислительной установке.

2) Прикладные программы пакета размещаются в разных узлах РВС. Исполнение схемы решения задачи выполняется в режиме интерпретации.

Последний подход характеризуется рядом потенциальных технологических решений по организации пакета, представляющих сложность при их реализации в рамках первого подхода:

- возможность разработки пакета для гетерогенной (многоплатформенной) вычислительной среды;
- допустимость включения в состав функционального наполнения пакета нетиражируемых прикладных программ, жестко привязанных к узлам РВС;
- повышение степени отказоустойчивости пакета при выполнении схемы решения задачи за счет введения вычислительной избыточности путем размещения всех прикладных программ этого пакета в каждом узле РВС.

В данной работе рассматриваются языковые средства описания предметной области пакета прикладных программ для РВС и особенности внутреннего представления этого описания, обусловленные процессом выполнения схем решения задач в режиме интерпретации. Основные методологические принципы описания предметной области для распределенного пакета прикладных программ приведены в работе [3].

2. Способы формирования схемы решения задачи

В рамках нашего подхода функциональное наполнение пакета, в целом, состоит из множества вычислительных модулей, размещенных в узлах РВС. Модуль реализуется в виде исполняемой в пакетном режиме программы, снабженной сцецификацей по ее назначению,

применению, времени выполнения, формату входных и выходных данных (параметров). Спецификации модулей определяют зависимости по входным и выходным параметрам. Множество спецификаций параметров и модулей формируют описание исследуемой предметной области.

Задание пользователя пакета представляется в виде схемы решения задачи, определяющей множества входных и выходных параметров задачи и порядок применения модулей, требуемых для нахождения искомых значений параметров по исходным данным. Таким образом, вычислительный процесс (процесс интерпретации схемы решения задачи) предполагает выполнения ряда зависимых между собой подзадач (модулей). В этом случае сложность управления таким процессом в РВС значительно возрастает [4].

В пакетах, как правило, реализуются два основных вида постановки задачи: процедурная и непроцедурная.

Планирование по непроцедурной постановке задачи вида «по заданным значениям параметров $x_1, x_2, \ldots, x_k$ вычислить значения параметров $y_1, y_2, \ldots, y_r$» позволяет автоматически получать схему решения задачи, не вдаваясь в понимание зависимостей между входящими в нее модулями. Такой способ построения схемы решения задачи подходит для пакетов с большим количеством параметров и модулей.

Однако пакет включает, зачастую, незначительное число модулей и предполагает строгую последовательность их выполнения. В этом случае постановка задачи в процедурном виде является более предпочтительной и обеспечивает пользователю возможность самостоятельно определять порядок вычислений в процессе решения задачи и использовать дополнительные конструкции языка управления заданием.

3. Входной язык

В качестве входного языка для описания модели предметной области пакета используется метаязык XML [5]. Метаязык предоставляет набор правил для создания собственного языка. Важной чертой XML является расширяемость (eXtensible), что позволяет совершенствовать и дополнять входной язык новыми средствами и возможностями без потери совместимости с предыдущими его версиями. Входной язык, разработанный на основе XML, позволяет различным распределенным системам обмениваться данными посредством общих структур и спецификаций. В силу ограниченности объема работы полное формальное описание входного языка не приводится, а его основные особенности иллюстрируются примерами спецификаций различных объектов предметной области.

Описание предметной области включает следующие основные секции: спецификации параметров, модулей и постановок задач. На Рис. 1 приведен фрагмент описания предметной области, включающий спецификации трех параметров, двух модулей и одной постановки задачи.

```
<!-- секция спецификаций параметров -->
<parameters>
   <param id='1' name='model' type='file' filename='model.txt'>
   <param id='2' name='model_list' type='filelist'
              pattern='model_element_%1.txt' size='10'/>
   <param id='3' name='result_list' type='filelist'
              pattern='result_element_%1.txt' size='10'/>
</parameters>
<!-- секция спецификаций модулей -->
<modules>
   <module id='1' name='decompose'>
         <commands os='Windows'>
               <start>decompose.exe (param_id:1)</start>
         </commands>
         <parameters>
               <input><param id='1'/></input>
               <output><param id='2'/></output>
         </parameters>
```

```
      </module>
      <module id='2' name='solve_decomposed_model'>
            <commands os='Linux'>
                  <start>./solver (param_id:2)</start>
            </commands>
            <parameters>
                  <input><param id='2'/></input>
                  <output><param id='3'/></output>
            </parameters>
      </module>
   </modules>
   <!-- секция спецификаций постановок задач -->
   <processes>
      <process>
            <stage>
                  <module id='1'/>
            </stage>
            <stage>
                  <module id='2' listParameterId='2'/>
            </stage>
      </process>
   </processes>
```

Рис. 1. Фрагмент описания предметной области на входном языке

В первой секции описываются все параметры, которые могут быть задействованы в процессе решения задачи. Каждый параметр снабжается набором атрибутов: уникальным идентификатором, концептуальным именем, типом и соответствующими типу дополнительными атрибутами. К допустимым типам параметров относятся *файл и список файлов*. Дополнительными атрибутами параметра типа *файл* являются имя файла для этого параметра на диске. Дополнительными атрибутами параметра типа *список файлов* являются число элементов списка и шаблон имен файлов для этих элементов на диске.

Секция <modules> содержит спецификации, определяющие назначение модуля, атрибуты (идентификатор, имя модуля), команды запуска, а также множество входных и выходных параметров.

В секции <processes> описываются различные схемы решения задач для заданной предметной области. Каждая схема задается в ярусной форме, определяющей последовательность выполнения модулей для вычисления требуемых параметров.

С целью повышения степени интеллектуализации средств управления вычислительным процессом в спецификацию схемы решения задачи заложена возможность включения управляющих конструкций для анализа текущего состояния вычислительного процесса и принятия решений о дальнейшем ходе вычислений. Применение таких конструкций основывается на системе событий, происходящих при обработке схемы решения задачи. К ним относятся:

- *beforeStart* – готовность модуля к запуску;
- *onFinish* – завершение работы модуля;
- *onFail* – некорректное завершение модуля;
- *onStop* – вынужденная остановка модуля;
- *onTimer* – мониторинг результатов вычислений, выполняемый с определенным интервалом.

Разработчик пакета может самостоятельно определять функциональное наполнение для обработки происходящих в процессе вычислений событий. Такой обработчик реализуется в виде набора операторов на языке QSA [6]. Язык QSA расширен программным интерфейсом DiscompAPI, позволяющий пользователю взаимодействовать с исполнительной подсистемой инструментального комплекса. Интерфейс DiscompAPI предоставляет следующие функциональные возможности: получение/изменение значений параметров, остановка/запуск процесса выполнения модуля (всех модулей на ярусе), передача управления на нужный ярус.

Корректность обработки событий средствами DiscompAPI обеспечивается ограничениями целостности, накладываемыми на отношения между объектами предметной области (параметрами, модулями, значениями параметров и др.) в процессе вычислений.

На Рис. 2 приведен пример описания условия остановки вычислений при обработке параметра типа список файлов. После завершения каждого экземпляра модуля *module 2*, запускается обработчик *checkListResult()*, который анализирует выходной элемент списка *param 3*. Если выходной элемент отличен от нуля, то производится остановка всех модулей на текущем ярусе и вычислительный процесс завершается. Применение данной конструкции позволяет избегать избыточных вычислений при решении комбинаторных задач большой размерности, основным назначением которых является поиск результата удовлетворяющего определенным условиям.

```
<process>
  <stage>
        <module id='1'/>
  </stage>
  <stage>
        <module id='2' listParameterId='2'
                        onFinish='checkListResult()'/>
  </stage>
</process>
<actions>
<![CDATA[
  function checkListResult() {
        var res = DiscompAPI.getModuleListOuputParameter(2,3);
        if ( res != 0 ) {
              DiscompAPI.stopCurrentStageModules();
        };
  };
]]>
</actions>
```

Рис. 2. Включение управляющих конструкций в схему решения задачи

Следующий пример (Рис. 3) иллюстрирует задание способа обработки события *onFinish* при параллельном запуске модулей, реализующих различные алгоритмы решения одной и той же задачи и идентичных по формату входных и выходных параметров. Применение подобной конструкции целесообразно в случае, если заведомо неизвестно какой из алгоритмов будет наиболее эффективным для вычисления значений выходных параметров.

```
<stage>
  <module id='1' onFinish='DiscompAPI.stopModules([2,3]);'/>
  <module id='2' onFinish='DiscompAPI.stopModules([1,3]);'/>
  <module id='3' onFinish='DiscompAPI.stopModules([1,2]);'/>
</stage>
```

Рис. 3. Пример задания способа обработки события onFinish

После завершения выполнения одного из модулей, происходит событие *onFinish*, обработчик которого производит остановку остальных модулей. При таком подходе выполнение избыточных вычислений, во-первых, обеспечивает решение задачи за минимальное время, во-вторых, увеличивает степень отказоустойчивости вычислительного процесса. Следует отметить, что данная постановка задачи эффективна только в том случае, если число доступных вычислительных узлов больше или равно количеству модулей.

4. Внутреннее представление

Транслятор инструментального комплекса DISCOMP переводит описание предметной области пакета во внутреннюю объектно-ориентированную структуру (Рис. 4) и осуществляет проверку целостности модели данной предметной области.

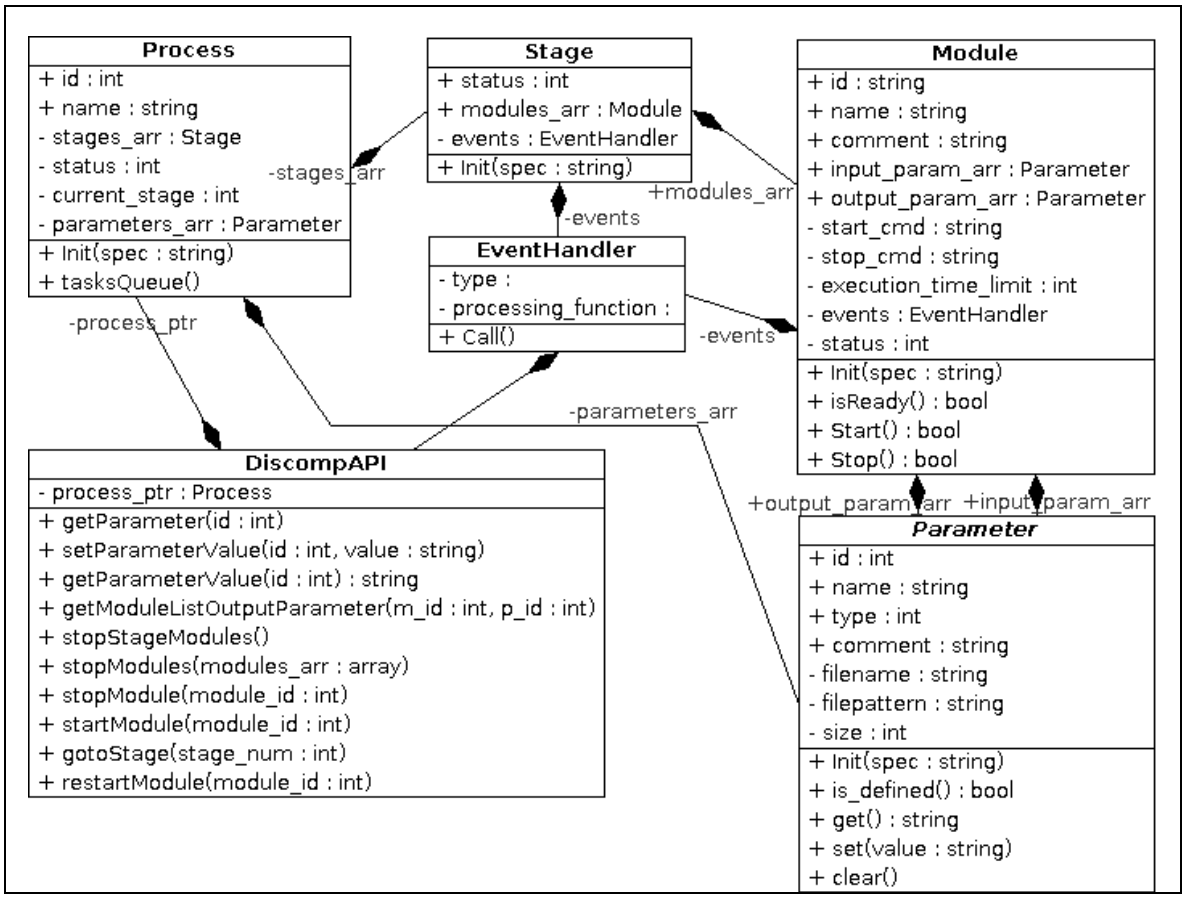

Рис. 4. UML-диаграмма классов объектов для внутреннего представления вычислительного процесса

Процесс трансляции во внутреннюю структуру включает в себя следующие этапы. Описание предметной области передается в качестве параметра методу *Init* объекта *Process*, который выполняет анализ всех структур, входящих в это описание, и производит разбор схемы решения задачи. Для каждого яруса схемы создается новый объект *Stage*, инициализируемый фрагментом спецификации. После инициализации каждый объект *Stage* добавляется в массив *stages_arr* объекта *Process*. Модули, которые должны быть выполнены на каждом ярусе, инициализируются по спецификациям входящим в предметную область и добавляются в массив *modules_arr* объекта *Stage*. Подобным образом инициализируются параметры для каждого из модулей.

Вычислительный процесс предполагает выполнение всех ярусов (элементов массива *stages_arr* объекта *Process*) для данной схемы решения задачи. При выполнении каждого яруса производится инициализация и запуск модулей на доступных узлах распределенной среды. После завершения всех модулей яруса (элементов массива *modules_arr* объекта *Stages*), производится переход к следующему ярусу и обработка схемы решения задачи продолжается.

5. Заключение

В заключение следует отметить следующие особенности представленного в данной работе подхода к разработке пакетов прикладных программ:

- представление пакетных знаний на метаязыке XML;
- построение схемы решения задачи на основе процедурной постановки задачи;
- выполнение вычислительного процесса в режиме интерпретации;
- предоставление конечному пользователю программных средств управления процессом вычислений.

На сегодняшний день инструментальный комплекс DISCOMP использован для создания

ряда пакетов прикладных программ в гетерогенной многоплатформенной РВС [7, 8, 9]. Схемы решения задач в этих пакетах формируются с использованием описанных в статье языковых средств, с применением различных конструкций управления вычислительным процессом.

Литература

1. Коваленко В.Н., Корягин Д.А. Вычислительная инфраструктура будущего // Открытые системы. – 1999г. . – № 11-12. – С. 45-53.
2. Воеводин Вл.В., Филамофитский М.П. Суперкомпьютер на выходные // Открытые системы. – 2003г. – № 5. – С. 48-56
3. Опарин Г.А., Феоктистов А.Г. Инструментальная распределенная вычислительная САТУРН-среда // Программные продукты и системы. – 2002. – №2. – С. 27-30.
4. Гершуни Д.С. Теоретические основы и методы проектирования распределенных систем / Вычислительная техника. Системы управления. – 1991. – В.6. – С.4-51.
5. Спенеср П. XML проектирование и реализация. – Издательство “Лори”, 2001. – с. 510.
6. Qt Script for Applications: [http://doc.trolltech.com/qsa-1.2.2/index.html], 31.10.2007.
7. Сидоров И.А., Феоктистов А.Г., Тятюшкин А.И. Распределенная информационно-вычислительная среда модульного программирования // Параллельные вычисления и задачи управления: Труды III Межд. конф. PACO’2006. – М.: ИПУ РАН, 2006. – С. 505-521.
8. Заикин О.С., Семенов А.А., Феоктистов А.Г., Сидоров И.А. Параллельная технология решения SAT-задач с применением пакета прикладных программ D-SAT // Вестник ТГУ. – Приложение. – 2007.– №23. – С. 83-95.
9. Опарин Г.А., Богданова В.Г., Сидоров И.А. Интеллектуальный решатель задач в булевых ограничениях в распределенной вычислительной среде // Информационные и математические технологии в науке и управлении: – Иркутск: ИСЭМ РАН, 2007. – С. 32-40.