Известия Гомельского государственного университета
имени Ф. Скорины, №5 (44), 2007

ИНФОРМАТИКА

УДК 519.8

Технология имитационного моделирования на платформе .NET с параллельным выполнением процессов

С. А. Альховик, Р. В. Петров

Широкое применение имитационного моделирования в ходе проектирования и эксплуатации сложных систем делает актуальной задачу создания инструментальных средств построения и исследования имитационных моделей (ИМ). Существующие системы имитационного моделирования малоэффективны при применении их для создания ИМ сложных многоуровневых иерархических систем. Требуется разработка новых технологий и средств имитации, используя последние достижения в области информационных технологий.

Использование платформы .NET, которая имеет в своем составе все необходимые средства для реализации управляющей программы моделирования (УПМ), позволяет существенно упростить процесс разработки ИМ. Например, используя возможность динамически анализировать структуру программы во время ее выполнения, можно исключить дополнительное, помимо программного модуля, описание компонентов и избежать дублирования в хранении информации. Средства управления памятью .NET значительно сокращают количество ошибок в программе ИМ, и тем самым – время на ее отладку. Платформа .NET не ограничивает разработчика использованием какого-то конкретного языка. В качестве языка, на котором реализована УПМ, выбран C#, однако разработка имитационной модели может вестись на любом .Net языке программирования.

ИМ представляется в виде совокупности взаимосвязанных процессов (Process) и хранилищ данных(Storage). Хранилища данных определяют состояние системы. Процессы описывают функциональные действия компонентов системы. Каждый процесс имеет набор полей, посредством которых обеспечивается связь с хранилищами.

Процессы в ИМ представлены экземплярами классов, производных от CustomProcess (рис. 1). Для описания полей процесса используется шаблонный класс Field<>. В качестве параметра шаблона указывается тип данных хранилища, к которому может подключаться поле. Соответствующие поля класса помечаются атрибутом FieldAttribute. В качестве параметров атрибута можно использовать: Name – наименование поля; Id – идентификатор, используемый при обращении к полю из другого процесса; IsReadOnly – указывает, что процесс не меняет значение поля во время своей работы. Если имя или идентификатор не указан, то используется имя поля класса. Параметр IsReadOnly по умолчанию имеет значение False.

```
public class MyFirstProcess : CustomProcess {
        [Field(Name=”Вход”)]
        Field<double> Input;
        [Field(Name=”Выход”)]
        Field<double> Output;
        public override void Run()    {
                for (;;) { Wait(1.0, delegate { return Input.Value > 0; });
                        Output.Value = Input.Value;
                        Input.Value = -Input.Value;   } }
}
```

Рисунок – 1. Пример описания процесса

Для реализации алгоритма процесса в классе процесса перегружается метод Run (см. рис. 1). Для синхронизации с другими процессами используется метод Wait. Этот метод может быть помещен в любом месте метода Run и служит для ожидания наступления заданного момента времени и/или состояния системы. Доступ процессов к общим хранилищам синхронизирован. При работе процесса хранилища из его окружения не могут быть изменены другими процессами. Изменения могут произойти только во время вызова процессом метода Wait.

Для создания модели системы используется класс, производный от класса CustomModel (рис. 2).

```
[DefaultModel]
public class MyFirstModel : CustomModel
{
        [StorageAttribute(Name=”Вход”)]
        Storage<double> Input;
        [Storage(Name=”Выход”)]
        Storage<double> Output;
        [Process]
        public MyFirstProcess FirstProcess;

        public override void ConstructModel(IObjectMaker objectMaker) {
                base.ConstructModel(objectMaker);
                MyFirstProcess.Attach(Input);
                Processes[“FirstProcess”].Fields[“Output”].Attach(“Output”);
} }
```

Рисунок – 2. Пример описания класса модели

В этом классе, аналогично полям процесса, описываются хранилища и процессы модели. Для этого используются класс Storage<> (для описания хранилищ) и разработанные ранее классы процессов, а также атрибуты StorageAttribute и ProcessAttribute (см. рис 2). Параметром шаблона Storage<> является тип данных хранилища.

При описании хранилища можно задать следующие характеристики: Name, Id, IsReadOnly – аналогично полям процесса; IsParameter – определяет, является ли данное хранилище параметром модели; IsResponce – определяет, является ли значение хранилища откликом модели. Установка параметров IsParameter и IsResponse в True (по умолчанию) приведет появление ссылок на это хранилище в соответствующих секциях файла данных модели.

При описании процесса используются параметры: Name – наименование процесса; Id – идентификатор процесса, используется для доступа к нему из других процессов; Priority – приоритет процесса. Параметры Name и Id, если не указаны, принимаются равными имени поля класса, которое описывает соответствующий процесс.

Для создания связей между полями процесса и хранилищами используется метод ConstructModel. В этом методе для каждого поля процесса вызывается метод Attach с параметром хранилище (см. рис. 2) В результате поля процессов будут соединены с заданными хранилищами. В качестве типов данных для полей и хранилищ может быть использован как любой встроенный тип, так и собственный класс. Для доступа к значению поля либо хранилища используется свойство Value.

Возможно динамическое создание и изменение структуры модели (перечня процессов и хранилищ и связей между ними) посредством методов AddProcess / RemoveProcess, AddStorage / RemoveStorage. Параметры этих методов совпадают с параметрами соответствующих атрибутов и дополнительно включают тип создаваемого процесса или хранилища.

Доступ к процессам и хранилищам модели, полям процессов осуществляется посредством коллекций Model.Processes, Model.Storages и Process.Fields соответственно.

Алгоритм цикла моделирования имеет следующий вид:

1. Перед началом моделирования создается объект класса модели и для него вызывается метод ConstructModel, в этом методе происходит создание всех процессов и хранилищ, а также связей между ними.

2. Вызывается метод Initialize и происходит создание потоков для всех процессов, процессы переходят в состояние ожидания своего флага «Запуск» и разблокирования нужных хранилищ в методе ThreadProc.

3. Выполняется цикл моделирования (пока модельное время меньше заданного) в этом цикле происходит вызов метода Activate модели.

3.1. После первого вызова метода Activate для процесса управление из метода ThreadProc передается в метод Run. Работа процесса продолжается до выхода из метода Run либо до вызова метода Wait.

3.2. После вызова метода Wait управление передается в метод WaitModel, а из него в управляющий алгоритм, до следующего вызова метода Activate.

4. Полная остановка модели осуществляется вызовом метода Terminate.

После создания класса модели запуск ИМ может быть осуществлен программным модулем Modeler.exe, который обеспечивает ввод исходных данных и вывод результатов моделирования посредством XML-файлов, что обеспечивает интеграцию с подсистемой планирования, проведения и обработки результатов имитационных экспериментов ПТКИ BelSim [1].

Каждый процесс модели реализуется как отдельный поток (Thread) операционной системы. Потоки между собой синхронизируются с помощью объектов событие (Event), для чего каждое хранилище данных имеет событие, позволяющее осуществлять к нему монопольный доступ, а также каждый процесс имеет два события-флага «Ожидание» и «Запуск». Синхронизация состоит из двух частей:

1. Каждый процесс находится в состоянии «Ожидание» и ожидает своей активизации (установка события-флага «Запуск»), а также ожидает разблокирования всех связанных с ним хранилищ. После этого управление передается в алгоритм процесса, который, выполнив нужные действия, вызывает метод Wait для синхронизации и процесс переходит в состояние «Ожидание».

2. Моделирующий алгоритм устанавливает для процессов, в зависимости от приоритета, флаг «Запуск» (перед началом запуска процессов, и после окончания их работы все хранилища разблокированы) и ожидает перехода всех процессов в состояние «Ожидание».

Для проверки эффективности рассмотренной выше реализации УПМ разработана тестовая модель, представляющая собой совокупность однотипных процессов, алгоритм которых состоит из операции с фиксированной трудоемкостью и оператора ожидания с некоторым постоянным интервалом времени. Параметрами модели являются трудоемкость операции в активности процесса и количество процессов. Время моделирования (и, соответственно, число активизаций каждого процесса) выбиралось достаточно большим, чтобы обеспечить возможность замера машинного времени работы модели, выступающего в качестве отклика. Результаты экспериментов на двухпроцессорной ЭВМ представлены на рис. 3, где T_o – расчетное время выполнения активности; T_{p2}, T_{s2} – среднее время выполнения активности при параллельном и последовательном выполнении процессов соответственно. Последовательная активизация процессов обеспечивалась назначением им приоритетов по номеру процесса в порядке возрастания.

Для двух процессов T_{p2} уменьшается в два раза, при этом, очевидно, алгоритмы процессов выполняются параллельно на разных процессорах. При трех процессах наблюдается некоторый скачок времени выполнения, которое с дальнейшим увеличением количества процессов уменьшается. Такое поведение, вероятно, обусловлено влиянием алгоритма планирования и диспетчеризации потоков выполнения операционной системы, доля времени выполнения которого постепенно начинает снижаться с ростом количества процессов в модели. Рост T_{s2} при увеличении количества процессов обусловлен присутствующей в алгоритме УМП операцией сортировки процессов по приоритету.

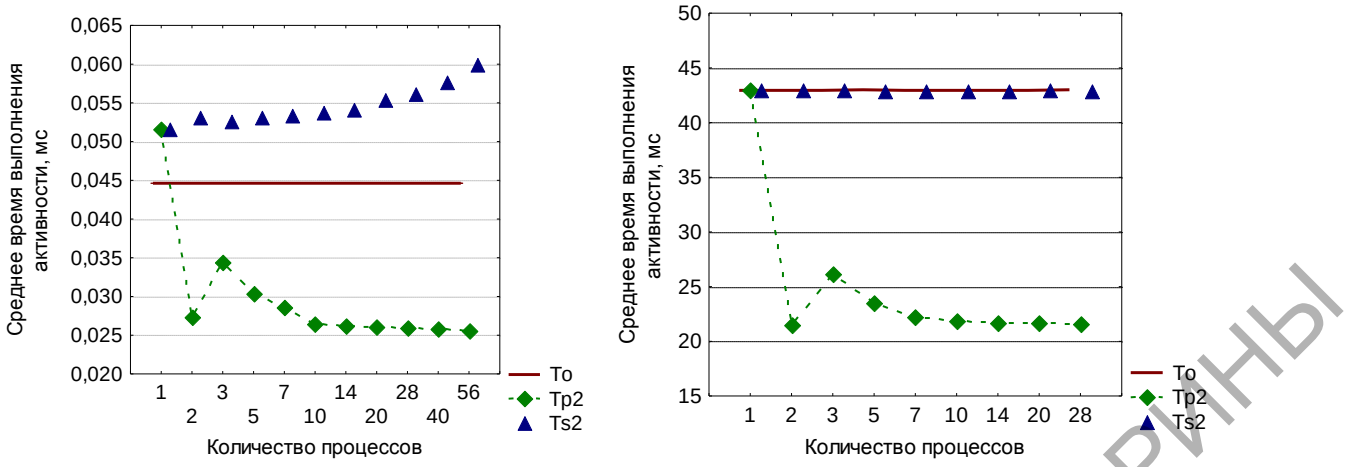

Рисунок – 3. Зависимость среднего времени выполнения активности процесса от количества процессов

Таким образом, путем выполнения алгоритма каждого процесса модели в отдельном потоке появляется возможность эффективно использовать ресурсы многопроцессорных ЭВМ, что особенно актуально при моделировании сложных многоуровневых иерархических систем. Представленная технология имитационного моделирования использована для создания ИМ технологического процесса производства полиэтилентерефталата на заводе органического синтеза ОАО «Химволокно» (г. Могилев) с целью решения задачи оптимизации переходного процесса при смене ассортимента.

Abstract. Simulation modeling technique using .NET framework with parallel process execution by means of multithreading that enables to use resources of multiprocessor computers is presented in the paper.

Литература

1. Якимов, А.И. Имитационное моделирование в ERP-системах управления / А.И. Якимов, С.А. Альховик // Минск: Бел. наука, 2005. – 197 с.: ил.

Белорусско-Российский университет Поступило 15.05.07