ISSN 0002-306X. Изв. НАН РА и ГИУА. Сер. ТН. 2002. Т. LV, №2.

УДК 681.3

ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА И ИНФОРМАТИКА

А. Л. БАЛАГЁЗЯН

ОБЪЕКТНО-ОРИЕНТИРОВАННЫЙ ПОДХОД К ПРОЕКТИРОВАНИЮ ПРОЦЕССОРА ВИРТУАЛЬНОЙ СРЕДЫ

Рассматривается применение объектно-ориентированного подхода к проектированию интерпретатора виртуальной среды. Предложена и обоснована структура процессора виртуальной среды и получены его временные характеристики.

Ключевые слова: процессор виртуальной среды, интерпретация процессов, временные характеристики, отладка.

1. Задачи проектирования распределенных виртуальных сред. В настоящее время существуют два ключевых аспекта развития технологий проектирования изделий на интегральных схемах (ИС):

- Увеличение степени интеграции, уменьшение размеров и увеличение быстродействия приводят к экспоненциальному увеличению сложности при каждом добавлении новых возможностей в изделие. Управление процессом проектирования таких изделий вручную становится крайне сложным, а время на рассмотрение вариаций процесса проектирования и обнаружения его дефектов резко уменьшается.
- Время разработки очередного поколения технологий проектирования быстро (примерно в 2 раза за последние 4 года) уменьшается, в то время как, одновременно, цены на изготовление изделий, содержащих новые возможности, так же быстро увеличиваются.

Для решения первой проблемы необходимо управлять процессом, основанным на моделях, связанных в реальном времени как по данным, так и по управлению. Для решения второй - необходимо автоматизировать, т.е. перевести на уровень программной системы процесс разработки технологий проектирования.

В 1994 г. P. Raulefs [1] разработал принципы построения объектно-ориентированных систем для решения указанных проблем и предложил специальную расширяемую инфраструктуру, называемую ***виртуальная фабрика*** (ВФ). ВФ - это система программных моделей и приложений, которые управляют работой распределенной сети реальных фабрик. Реализация ВФ приводит к разработке автоматизированной системы с распределенными, параллельными архитектурами, разнородными операционными системами и разными аппаратными платформами.

Введение новой инфраструктуры обусловлено следующим.

Интеграция только по данным для систем управления процессами не позволяет легко модифицировать и расширять систему. В частности, может возникнуть необходимость заплат в приложениях, прямо не связанных с данной модификацией.

Применение объектно-ориентированного подхода (ООП) упрощает решение вопросов надежности, гибкости, расширяемости, быстрого внесения изменений, многократного использования путем создания общего каркаса для моделей распределенных систем, в которых объекты и процессы являются примитивами.

В [2, 3] исследована базовая модель виртуальной среды, предложенная в [1]. Введены новые понятия и алгоритмы, расширяющие эту модель. На базе полученных результатов, а также результатов по верификации процессов, описанных в [4], построена специальная объектно-ориентированная оболочка ***PB*** – *Process Builder*, которая позволяет формально описать процесс, выполнить его верификацию, применить оптимизирующие преобразования, а также интерпретацию и отладку процессов.

В настоящей работе рассматривается реализация интерпретатора процессов в среде ***PB***. Обоснована и описана архитектура соответствующей виртуальной машины [5] и получены ее временные характеристики. Описана также структура соответствующего программного обеспечения.

2. Базовая модель виртуальной среды. Рассмотрим расширенную модель, описанную в [2, 3]:

Object	=	**ID x StateSet x OperationSet**
State	=	**VariableValue**
Operation	=	**(State x Msg x Env) x (State x MsgSet x Env)**
Env	=	**ObjectSet**
Msg	=	**ValueList**
Behaviour	=	**(Msg x Env) x (MsgSet x Env)**

Объект (Object) состоит из *уникального идентификатора* ***(Unique Identifier)***, *множества состояний* ***(Set of States)*** *и множества операций* ***(OperationSet).*** *Операции* ***(Operation)*** выполняются в ответ на сообщения ***(Messages).***

Поведение объекта ***(Behavior)*** – это отношение, описывающее реакцию объекта при поступлении сообщения в зависимости от состояния и операции. Изменение состояния объекта после выполнения операции может привести к изменению его дальнейшего поведения.

Ситуация описывает ограничения на возможные состояния объектов в течение некоторого периода времени. Она состоит из ***утверждения о состояниях*** *(Assertion)* объектов и ***временного ограничения*** *(Time Constraint),* описывающего множество моментов времени, в течение которых рассматриваемое утверждение истинно. Более точно:

Situation = Assertion x TimeConstraint, где **Assertion: StateSet → {TRUE, FLASE}, Assertion(s) = TRUE**, если объект находится в состоянии **s,** и **FALSE** в противном случае. Т.е. каждое утверждение рассматривается как допустимые множества состояний одного объекта. **TimeConstraint = {t | (n ∈{0,1,…},** где **$t = t_0 + n*\Delta t$}.** Временное ограничение задается в виде линейной функции, которая описывает множество допустимых (периодических) моментов времени. Множество ситуаций, рассматриваемых как единое целое, обозначим как **пакет ситуаций**. **Процесс** определяется как сеть пакетов ситуаций и операций**. Операция** описывает механизм

изменения состояния соответствующего объекта при поступлении сообщения. Для каждой операции определено время ее выполнения. Каждый процесс начинается из одной вершины (*начальная вершина*) и заканчивается также в одной вершине (*конечная вершина*).

Ограничим теперь структуру сети, определив ее как специальную древовидную структуру, задающую последовательности ситуаций и операций, упорядоченных по временным ограничениям:

- Конечная вершина Z представляет собой особую конструкцию, которая не связана ни с каким объектом или ситуацией и имеет возможность перенаправлять посланные сообщения на начальную вершину. Она состоит из единственной ситуации особого объекта X, используемого для накапливания сообщений, которые дошли до конечной вершины, и переадресовки их после завершения процесса к объектам, присутствующим в начальной вершине.
- Каждый переход (Transition) в сети представлен операцией, выходящей из ситуации данной вершины и направленной на определенную вершину. Путь определяется списком переходов, следующих друг за другом.
- Каждая ситуация имеет не больше одной выходящей операции.
- В сети не существует путей, содержащих повторяющиеся вершины (циклы).
- Для каждой ситуации в данной вершине сумма значения произвольного допустимого момента времени и длительности последующей операции не должна превышать соответствующего ограничения по времени для любой ситуации, содержащейся в вершине, к которой совершается переход.

Выполнение процесса соответствует моменту преобразования сети из ситуаций и операций в сеть из событий и операций, где вместо ограничений на состояния и время в ситуациях предоставляются конкретные события. Результирующее множество сетей событий - операций называется ***историей процесса (Process History).***

Процесс задается по условиям извне, которые для операций объектов индуцируют соответствующие им события начальной ситуации. События, в свою очередь, порождают новые события и т.д.

Краткое описание семантики процессов приведено в [3]. Основываясь на этом описании, а также обозначениях, использованных в той же работе, приведем более подробное описание, необходимое для изложения наших результатов.

Введем следующие обозначения:

- **Input Scene** = (t_0 , $msgs_0$, $States_0$) – входная сцена.
- **RRSS**(*Ready to Run Situation Set*) – множество ситуаций, готовых к выполнению.

ESS(*Executory Situation Set*) – сообщение получено, Assertion(current state)=T, но текущее время ∉ TimeConsrtaint.

- **RSS**(*Running Situation Set*) – множество выполняющихся ситуаций.
- **Output Scene** = (t , msgs , States) – выходная сцена.

SitPredicate(s,t) = Assertion(s) & t ∈ TimeConstraint – предикат готовности.

Выполнение процесса формально выглядит следующим образом [2]:

0. RRSS = ∅; ESS = ∅; RSS = ∅;
1. RRSS : = Situations(StartNode);
   для всех sit, s вместе с [sit∈ RRSS & s ∈ sit & Assertion(s) = TRUE & "объект находится в состоянии s и получил сообщение msg"]
   выполнять
2. { ESS : = ESS U { sit};
   RRSS : = RRSS \ { sit };
3. если RSS = ∅ & ESS = ∅, то перейти к шагу 6;
4. пока sit ∈ ESS & SitPredicate(s,t) = TRUE выполнять
   RSS : = RSS U { sit };
   ESS : = ESS \ { sit };
5. пока sit ∈ RSS & Sit_Trans_Event (s) = TRUE выполнять
   RSS : = RSS \ { sit };
   если NextNode(sit) ≠EndNode и NextNode(sit).Transition > 0 то
   RRSS : = RRSS U Situations(NextNode(sit)); }
6. конец

3. Процессор виртуальной машины. Структура и компоненты ПВС. Построим процессор виртуальной системы (ПВС) интерпретации процессов, которая интерпретирует введенную модель процесса по аналогии с процессором, интерпретирующим обычные последовательные программы [5, 6]. Рассмотрим базовую структуру интерпретатора последовательных программ (рис.1), реализующую следующий алгоритм:

1. Инициализация, запись в память программы и данных, установка счетчика команд.
2. Выборка очередной команды из памяти и увеличение счетчика команд.
3. Декодирование команды и выборка операндов.
4. Выполнение команды и запись результатов в память.
5. Переход к шагу 2.

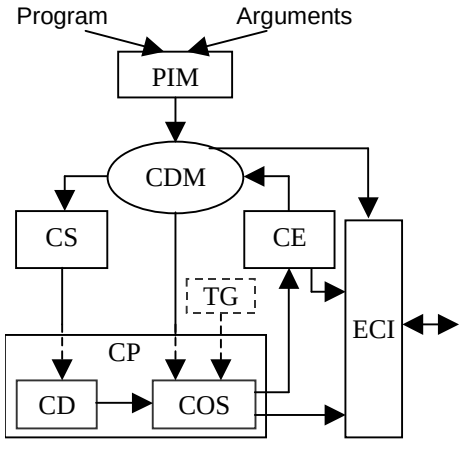

Рис. 1. Структура процессора

- Program Initialization Module (PIM) – блок записи исходной программы в память и установки счетчика команд.
- Command and Data Memory (CDM) – память, содержащая программу и данные.
- Command Selector (CS) - блок выборки команды из CDM.
- Command Decoder (CD) – блок декодирования команды.
- Command Operands Selector (COS) – блок выборки операндов команды из CDM.
- Command Executor (CE) – блок выполнения команды.
- Command Pipe (CP) – конвейер команд.
- External Communications Interface (ECI) – интерфейс для общения процессора с внешней средой (для подключения внешних устройств).

Обозначим время выполнения шага i приведенного алгоритма через t_i. Тогда время выполнения одной команды равно $Tc = t_2 + t_3 + t_4$, а общее время выполнения последовательной программы - $Tp = Tc \cdot Cc$ ·Cc – количество команд программы).

Между выполнением последовательной программы и рассматриваемой объектно-ориентированной модели процессов существует естественное соответствие, основанное на аналогах команды и ситуации. Единственным отличием здесь является то, что при выполнении процесса значение времени используется для определения готовности ситуации, а для последовательных программ таких параметров (операндов) нет. Это приводит к добавлению специального модуля *Time Generator - TG* в структуру процессора. Функцией TG является посылка синхросигнала всем блокам через заданный интервал времени и текущего значения таймера блоку COS. Интервал обновления таймера вычисляется в блоке PIM для каждого процесса и является числом, которому кратны все интервалы истинности ситуаций (TimeConstraints) в процессе.

3.1. Внутренние структуры данных ПВС и базовые операции над структурами данных. Базовый процессор состоит из следующих частей:

1. Process Network (PN) – память, содержащая исходный процесс в виде списка.
2. Object State Set (OSS) – память, содержащая состояния объектов в виде списка.
3. Message Set (MS) – память с накоплением, содержащая сообщения в виде списка.
4. Process Converter Module (PCM) – блок анализа и конвертации процесса в список (PN).
5. Process Walker (PW) – блок выборки очередной ситуации из PN для последующей обработки.
6. Time Generator (TG) – тактовый генератор.
7. Situation Processing Module (SPM) – блок выполнения операции.
8. Ready to Run Situation Set (RPSS) – память, где хранятся готовые к выполнению ситуации (ожидающие сообщения) в виде списка.
9. Executory Situation Set (ESS) – память, где хранятся готовые к выполнению ситуации (получившие сообщение, ожидающие времени выполнения).
10. Running Situation Set (RSS) – память, где хранятся выполняющиеся ситуации в виде списка.
11. Message Comparator (MC) – блок сравнения сообщений.
12. Time Comparator (TC) – блок сравнения времени.

Они взаимосвязаны между собой, как показано на рис. 2:

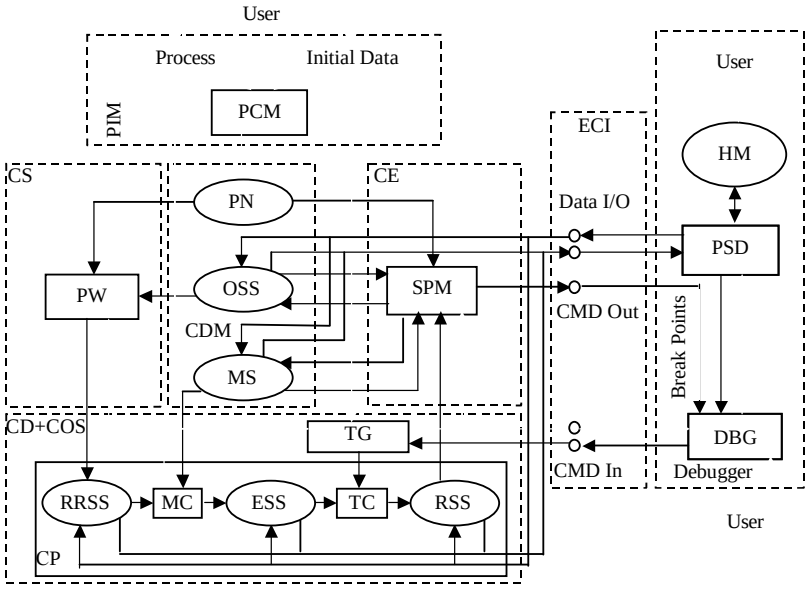

Рис. 2: а – базовая структура процессора, б - отладчик

3.2. Описание работы процессора. Схема работы процессора:

1. На вход PCM подаются процесс и начальные события. PCM конвертирует описание процесса в список и записывает в PN. На основе начальных событий выбираются начальные состояния объектов для OSS, сообщения для MS и вычисляется минимальный интервал времени для TG. Все данные записываются соответственно в OSS, MS и TG. На CMD In подается команда Run.
2. PW очищает память RRSS, считывает текущие состояния объектов и осуществляет поиск ситуаций, готовых к выполнению в PN. При нахождении записывает эти ситуации в RRSS.
3. MC считывает очередное сообщение и производит поиск соответствующей ситуации в RRSS, при нахождении ситуация перемещается в ESS. Поиск продолжается до тех пор, пока список ситуаций не будет пройден от начала до конца один раз или $MC \neq \varnothing$. Поиск возобновляется при изменении содержимого RRSS или MS.
4. Если $ESS \neq \varnothing$, то при каждом поступлении обновления от TG осуществляется поиск ситуации в ESS. Если обнаружена ситуация, для которой TimeConstraint = CurrentTime, то эта ситуация перемещается в память RSS.
5. SPM последовательно считывает ситуации из RSS и выполняет связанную с ней операцию. После считывания очередной ситуации в PN производится поиск перехода и извлечение операции. Далее из OSS и MC соответственно считывается состояние и сообщение объекта, связанного с ситуацией. Выполняется операция, сообщения и новые состояния некоторых объектов записываются в MC и OSS соответственно. Работа SPM продолжается до тех пор, пока $RSS \neq \varnothing$ & $OSS \neq \varnothing$ & $MS \neq \varnothing$.
6. Переход к шагу 2.

Блоки PW, MC, TC, TG, SPM могут работать параллельно и независимо друг от друга.

Для соединения с внешними устройствами введем в состав процессора 4 порта ECI (рис. 2 а). К этим портам можно подключать другой процессор, отладчик (рис. 2 б) и т. д.

Назначение этих портов следующее:

- **Data I/O –** предоставляет доступ ко всем фрагментам памяти, используемой в процессоре. Порт доступен по записи и чтению.
- **Sync I/O –** является выходом или входом синхронизирующего сигнала. Режим этого порта можно установить с помощью команд *Set Master Mode* и *Set Slave Mode.* Этот порт позволяет управлять работой внешних устройств или процессором с внешнего устройства.
- **CMD In –** порт, по которому можно подавать на процессор команды управления (список команд приведен ниже). Он доступен только по записи.
- **CMD Out –** порт, по которому процессор подает команды для управления внешним устройством. Он доступен по чтению.

Далее рассматривается новый компонент процессора – отладчик.

3.3. Пример использования интерфейса с внешней средой ECI. Механизм управления отладкой ПВС. Для добавления к процессору средств отладки опишем новое устройство (*Debugger*), которое подключается к процессору через ECI (рис. 2 б). Отладчик управляется синхронизирующими сигналами, поступающими от процессора, т. е. процессор должен работать в режиме *Master.* Введем следующие обозначения новых блоков:

1. History Memory (HM) – память (список) для накапливания и хранения состояний процессора.
2. Processor State Dump (PSM) – модуль сохранения состояния процессора в HM.
3. Debugger (DBG) - модуль управления процессором в режиме отладки.

Список команд данного процессора:

Команды порта CMD In:	Команды порта CMD Out:
Set Break Point (Data, Command, Time)	*Breakpoint Stop*
History Control	*Final Stop*
Restore State	*Fault Stop*
Set Mode (Master, Slave)	*Step Stop*

3.4. Временные характеристики ПВС. Оценим время работы одного цикла процессора. Для этого введем следующие обозначения. Пусть T_{mod} – время работы модуля процессора (где mod = PW, SPM, MC, TC). Тогда время выполнения одного цикла ограничено величиной $Tc = T_{PW} + T_{MC} + T_{TC} + T_{SPM}$, а общее время выполнения процесса -величиной $Tp = Tc \cdot Sc$ (Sc – количество ситуаций процесса).

Предположив, что время работы отдельного модуля кратно длине обрабатываемого списка, определим время работы одного модуля процессора. Пусть t_c - время операции сравнения, t_{ex} - время выборки очередного элемента списка из

памяти, t_{px} - время выборки очередного элемента по указателю (где $t_{px} < t_{ex}$), n_{PN} - количество ситуаций в PN, t_{PW} - время, за которое PW пройдет весь список PN. Тогда время обхода всего списка для модуля будет ограничено следующим выражением: $t_{PW} = (t_{ex} + t_c) \cdot n_{PN}$, аналогично можно определить время обхода для остальных модулей: $t_{MC} = (t_{ex} + t_c) \cdot n_{RRSS}$, $t_{TC} = (t_{ex} + t_c) \cdot n_{ESS}$, $t_{SPM} = (t_{ex} + t_c) \cdot n_{RSS}$, где t_c – время выполнения операции. Графически время работы отдельных блоков (для примера PW) и всего процессора выглядит следующим образом:

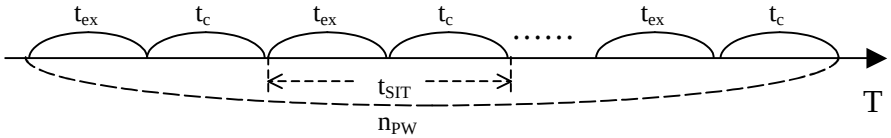

Рис. 3. Время работы блока PW, где t_{SIT} – время обработки 1-й ситуации

Утверждение 1.

Объекты PW, MC, TC, SPM могут работать с совмещением во времени за счет смещения на величину t_{SIT} влево по оси времени, моментов начала работы объектов MC, TC, SPM.

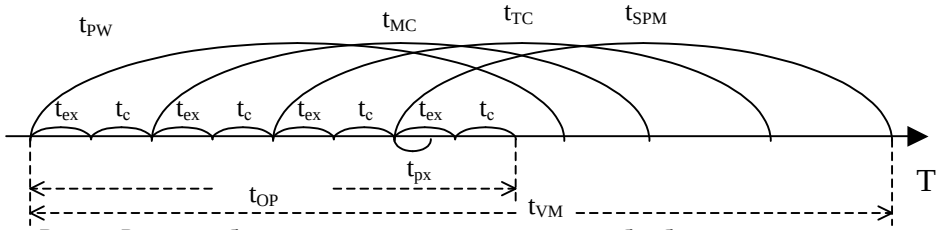

Рис. 4. Время работы процессора, где t_{OP} – время обработки 1-й операции, t_{VM} – время работы процессора

Доказательство следует из того, что в процессоре каждый объект PW, MC, TC, SPM работает с памятью, в которой данные для обработки хранятся списком и не имеют непосредственного взаимодействия друг на друга.

Утверждение 2.

Минимальная величина смещения вправо по отношению к началу работы объекта PW моментов начала работы объектов MC, TC, SPM составляет t_{SIT} для MC, $2 \cdot t_{SIT}$ для TC и $3 \cdot t_{SIT}$ для SPM.

Следующий уровень оптимизации требует внесения изменений в базовую структуру процессора. В частности, для уменьшения времени обработки одной ситуации одним модулем (t_{SIT}) необходимо добавить систему кэширования данных, а для ускорения обработки списков - увеличить количество модулей обработки ситуаций (например; 2 PW, 3 MC и т.д.).

4. Построение виртуальной среды на базе ПВС. Структура классов интерпретатора процессов. Выполним упрощения в структуре ПВС, связанные с выделением похожих частей в общий обрабатывающий модуль.

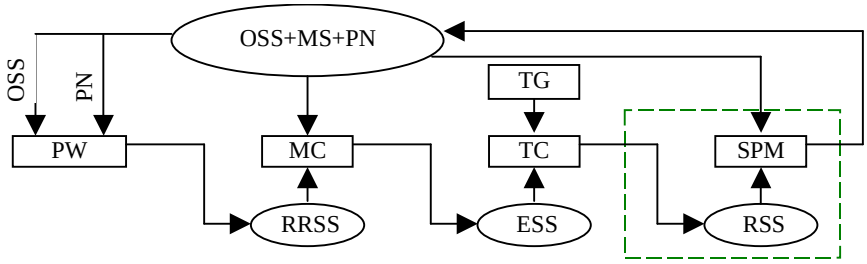

Рис 5. Упрощенная структура процессора

В настоящее время разработана и находится в стадии тестирования в корпорации Intel программная система **Process Builder,** ядром которой является описанный процессор ВС.

СПИСОК ЛИТЕРАТУРЫ

1. **Raulefs P.** The Virtual Factory // IFIP World Computer Congress 94. – 1994. - V.2. - P.18-30.
2. **Raulefs P., Shoukourian S., Grigoryan A.** Transformation of Hammock Type Processes // Proceedings of HPC‘2002, SCS International Advanced Simulation Technologies Conference ASTC’2002, USA - April 2002.
3. **Shoukourian S., Avagyan A., Tavangarian D.** Combination of Separate Processes in a Distributed Environment. A case of study // Proceedings of HPC‘2000, SCS International Advanced Simulation Technologies Conference ASTC’2000, USA - April 2000 - P. 280-285.
4. **Shoukourian S.** A Unified Design Methodology for Off-line and On-line Testing // IEEE Design and Test of Computers - April-June 1998 - P. 73-79.
5. **Terrence W. Pratt** Programming Languages Design and Implementation // Prentice Hall, 1996.
6. **Hennessy J., Patterson D.** Computer Architecture: Aquantitative Approach // Second Edition, M. Kaufmann Publ., 1995.

ГИУА. Материал поступил в редакцию 10.04.2002.

Ա. Լ. ԲԱԼԱԳՅՈԶՅԱՆ
ՕԲՅԵԿՏԱԿՈՂՄՆՈՐՈՇՎԱԾ ՄՈՏԵՑՄԱՆ ԿԻՐԱՌՈՒՄԸ ՎԻՐՏՈՒԱԼ ՄԻՋԱՎԱՅՐԻ ՊՐՈՑԵՍՈՐԸ ՆԱԽԱԳԾԵԼԻՍ

Դիտարկվում է օբյեկտակողմնորոշված մոտեցման կիրառումը վիրտուալ միջավայրի ինտերպրետատորը նախագծելիս: Առաջարկված և հիմնավորված է վիրտուալ միջավայրի պրոցեսորի կառուցվածքը, ստացված են նաև վերջինիս ժամանակային բնութագրերը:

A. L. BALAGYOZYAN
OBJECT-ORIENTED APPROACH TO DESIGNING VIRTUAL ENVIRONMENT PROCESSOR

Describes the application of object-oriented approach to designing virtual environment interpreter. The virtual environment processor is proposed and validated, and its time characteristics are obtained.