Методика иерархического исследования сложных дискретных структур

06, июнь 2012

DOI: 10.7463/0612.0370230

Рудаков И. В.

УДК 681.31

Россия, МГТУ им. Н.Э. Баумана

irudakov@yandex.ru

В настоящее время задача анализа сложных дискретных устройств является актуальной и находит большое количество применений. Однако, в силу исключительного разнообразия встречающихся на практике задач и недостаточной изученности их математического описания, арсенал формализации и моделирования непрерывно пополняется. Поэтому порядок структуризации, разбиения на подсистемы и элементы объектов и отбор математических схем для описания элементов нельзя считать окончательно сложившимся.

Известно [1, 2, 3], что при анализе и проектировании структур сложных систем используется блочно-иерархический метод, который предусматривает расчленение процесса проектирования на ряд последовательных уровней и сведения задачи большей размерности к совокупности задач значительно меньшей размерности.

Одной из задач при разработке структур сложных систем является задача анализа и контроля правильности функционирования системы на ранних этапах ее разработки, в результате чего определяется состав функциональных блоков и их взаимодействие. На последующих этапах осуществляется анализ отдельных блоков. Предлагаемый подход позволяет осуществить анализ функционирования сложной структуры с использованием методов макро моделирования и декомпозиции на базе языка моделирования ориентированного на транзакты.

Для реализации макро моделирования в качестве элементов моделирования предлагается использовать макро модели функциональных блоков сложной системы на основе формализации их функционирования в виде логической сети[4, 5].

Метод анализа сложных структур только на базе макро моделей функциональных блоков недостаточен, так как не позволяет учитывать такие характеристики системы как, например, временные параметры элементов, входящих в макро модель, а, следовательно, выполнять надежную верификацию проекта. С целью уточнения характеристик сложной системы

используется принцип “увеличительного стекла” [6, 7], когда различные участки структуры системы анализируются с разной степенью детализации. В отличие от традиционного подхода, при котором переход с одного иерархического уровня на другой выполняется разработчиком системы, предлагаемый метод декомпозиции, реализованный в виде программной подсистемы, позволяет выполнить автоматический переход от макромоделей функционального блока к моделям элементов структуры при выполнении смешанного моделирования сложных систем.

При описании макромоделей функционального блока как иерархической системы в виде логической сети необходимо учитывать:

- множество входных и выходных воздействий, поступающих на блок в соответствии с алгоритмом функционирования – $PS(P,T)$ и множество выходных воздействий, вызываемых блоком после реализации алгоритма – $QS(P,T)$ с учетом вероятностного (P) и временного характера параметров блока;

- множество связей между макромоделью функционального блока и моделями элементов $VS(P,T)$ с учетом множества обратных связей $WS(P,T)$;

- множество состояний макромоделей функционального блока – $F(P,T)$.

Практическая реализация иерархического исследования сложных дискретных структур опирается на технологию объектно-ориентированного программирования (ОПП). В качестве инструментального средства анализа и верификации структуры разработан язык моделирования на базе Visual C++, ориентированный на транзакты (ВИСМО).

Программное обеспечение иерархического проектирования структур с очередями функционирует в рамках ОС Windows XP и выше.

Для формализации процесса функционирования сложного дискретного устройства в виде логической сети в графическом виде введена следующая классификация блоков:

1. блок GENERATE – создание транзактов;
2. блок TERMINATE – уничтожение транзактов;
3. блок DEVICE – обслуживание транзактов;
4. блок QUEUE – организация очередей транзактов;
5. блок BRANCH – ветвление маршрута движения транзакт
6. блок COLLECT – объединение маршрутов движения транзактов.

Сеть моделируется при использовании механизма планирования событий (событийный подход). Считается, что изменения системы происходят в некоторые дискретные моменты времени – моменты наступления событий, а изменение состояния происходит мгновенно. Состояние системы характеризуется совокупностью переменных, причем начальное состояние устанавливается путем задания значений этих переменных. События, возникающие в модели, связаны с каждым блоком в отдельности. В некоторых случаях возникновение события характеризуется интервалом времени, через которое оно должно произойти, и планируется заранее (в моменты определения интервала времени). К такому виду событий относятся

события создания (блок GENERATE) и обслуживания (блок DEVICE) транзактов. В остальных случаях явно задать время возникновения событий не возможно, поэтому их наступление описывается с помощью условий (выход транзакта из блока QUEUE). Наличие условных событий приводит к тому, что помимо использования техники планирования событий необходимо использовать технику сканирования активностей. Учитывая то, что условные события связаны с состоянием системы (блок свободен/занят и т.д.), то и проверять условия срабатывания нужно только в моменты изменения состояния.

Управляющие алгоритмы опираются на специальные структуры: списки будущих, текущих и условных событий. Данные списки хранят информацию о планируемых событиях в системе. Методика хранения и индексирования событий и механизмы функционирования схематически изображены на рис. 1.

Список будущих событий постоянно поддерживается в отсортированном по возрастанию времени виде. Первый связанный список описывает ближайшие по времени события. В моменты протяжки модельного времени он связан с моментом времени, на которое в следующий момент переведется модельное время. В остальных случаях он связан с событиями, которые необходимо исполнить в текущий момент времени.

Список условных событий – это набор пар вида: событие и условие его срабатывания. Условие характеризует моменты наступления события. Как только выполняется условие, возникает соответствующее событие.

Рисунок 1. Структурная организация хранения событий и механизмы управления

Для оптимизации доступа к этому списку вводится дополнительный массив. С каждым элементом этого массива связан определенный параметр моделируемой системы, от которого может зависеть условие возникновения события и список индексов. Элементы такого списка являются ссылками на события, на которые может повлиять изменение связанного параметра.

Массив параметров формируется перед началом моделирования, причем изменения происходят в связанных индексных списках при добавлении и выборки событий из списка условных событий. В массив параметров первым элементом добавляется индексный список событий, условия срабатывания которых не удалось разбить по зависящим параметрам. Этот список хранит ссылки на события, условия которых проверяются каждый раз.

Для быстрого поиска индексов, которые нужно удалить при выборке событий из списка условных событий, к паре событие и условие добавляется третий элемент, который ссылается на элементы индексных списков.

Таким образом, планируемые события (привязанные к времени срабатывания) группируются по времени возникновения и упорядочиваются во времени (см. рис. 1) в специальном списке будущих событий. Модельное время изменяется согласно обработке событий из этого списка. Событие на обработку берется из начала списка. После обработки каждого события такого события, состояние системы может измениться, и по этой причине мы вынуждены проверить условия возникновения непланируемых событий. При срабатывании условия выполняется соответствующая обработка. Алгоритм функционирования заключается в последовательной обработке событий, попеременно сочетая обработку событий из списка будущих событий и проверку условий срабатывания непланируемых событий.

При разработке программного комплекса был выделен ряд основных объектов, которые в дальнейшем были взяты за основу при построении структуры классов (блок, оператор, исполняемый модуль, таблица и т.д.).

При построении программного комплекса была разработана модульная структура, изображенная на рис. 2, состоящая из трех частей:

- статическая библиотека (Values.lib), которая содержит описание такого понятия, как значение, и некоторые функции для работы с ним; данная библиотека подключается ко всем динамическим, используемым для расширения возможностей языка;

- системная динамическая библиотека (SysLib.dll), которая содержит описание базовых функций для работы со значениями, к таким функциям относятся функции сравнения значений, базовые арифметические функции (сложение, вычитание, умножение, деление), функции преобразования и функции генерации случайных величин;

- основной модуль (NetMassServ.exe), который несет на себе всю функциональную нагрузку, содержит описание объектов сети, алгоритмов функционирования этих объектов, поведение комплекса при редактировании и моделировании сети.

Рисунок 2. Модульная структура программного комплекса

Как уже было сказано, данная система моделирования имеет внутренний язык, с помощью которого происходит вычисление параметров блоков в процессе моделирования и обработка событий разомкнутой сети массового обслуживания. Основные понятия языка отображены на рис. 3.

Рисунок 3. Основные понятия языка

Ключевыми понятиями языка моделирования являются тип данных, выражения и операторы.

Тип данных определяет множество значений, характерного для параметров данного типа и множество операций. Указание типа данных производится для параметров создаваемых транзактов, переменных модели и блоков, локальных переменных. Указание типа происходит с помощью визуальных средств моделирования за исключением последнего случая, в котором тип указывается с помощью синтаксической конструкции языка.

Исполняемые элементы бывают следующих типов: это выражения и операторные блоки. Операторный блок – это набор операторов, исполняемый в момент возникновения какого-либо события, и является дополнительным механизмом обработки события.

Синтаксис языка описывается по следующим правилам:

```
<блок операторов> ::= <объявление локальных переменных> <набор операторов>
<объявление локальных переменных> ::= <пусто> | <объявление
переменной>;<объявление локальных переменных>
<объявление переменной> ::= <тип переменной> <имя переменной>
<выражение> ::= <константа> | <переменная> | <параметр транзакта> | <вызов
функции>
```

```
<вызов функции> ::= @<имя функции>(<параметры>) | @<имя функции>()
<параметры> ::= <выражение> | <выражение>,<параметры>
```

```
<оператор вызова функции> ::= <вызов функции>
<оператор присваивания значения> ::= <ссылка на значение> = <выражение>
<ссылка на значение> ::= <переменная> | <параметр транзакта>
<оператор условия> ::= if (<выражение>) <оператор>
<оператор цикла> ::= while (<выражение>) <оператор>
<составной оператор> ::= { <набор операторов> }
<набор операторов> ::= <пусто> | <оператор>;<набор операторов>;
<оператор работы с гистограммой> ::= #TABLE <номер таблицы>, <выражение>
<пустой оператор> ::= <пусто>
```

Данный программный комплекс имеет возможность подключения внешних динамических библиотек, что позволяет достаточно просто расширять возможности языка.

Разработанная библиотека имеет определенный формат, что требует выполнения следующих ограничений:

1. Библиотека обязательно должна содержать следующие функции:
 - LONG GetFunctionsList(
char** aStrList, // Массив имен функций
LONG iCount) // Максимальный размер массива

Функция через параметр `aStrList` возвращает список имен экспортируемых функций, `iCount` – максимальный размер массива `aStrList`. Функция возвращает число доступных функций в библиотеке.

- `CHECK_RESULT CheckFunction(`
`const CString& sFuncName, // Имя функции`
`VALUE_TYPE& vtRetType, // Тип возвращ. значения`
`VALUE_TYPE* vtParamsType, // Массив типов значений параметров,`
`передаваемых в функцию`
`LONG iParamCount, // Число параметров`
`CString& sErrorStr) // Строка с ошибкой`

Функция производит контроль вызовов функции `sFuncName` с параметрами `vtParamsType`, число которых `iParamCount`, и возвращает в `vtRetType` тип значения функции и строку с ошибкой `sErrorStr`, если она была.

- `void DeleteBlock(void* pBlock)`
Функция удаляет блок памяти `pBlock`, выделенный внутри библиотеки.

2. Подключаемые функции должны иметь следующий прототип:

```
typedef LPValue (*TExecuteFunction) (const CValueArray& aParams  
//Массив параметров);
```

3. Все вышеописанные функции должны быть экспортируемыми.

Рассмотрим практическое использование предложенного подхода на следующем примере. Требуется смоделировать автомобильный поток на участке путепровода и определить пожаробезопасные зоны. В качестве объекта исследований был выбран путепровод туннельного типа.

Данный путепровод имеет по три полосы движения в каждую сторону. Каждая полоса разбита на пожаробезопасные зоны (количество зон для каждой полосы соответственно: 6, 6, 5, 5, 4, 4). В случае возникновения чрезвычайной ситуации (пожара) происходит ее локализация с помощью специальных жароустойчивых штор, расположенных на границе зон. По путепроводу двигаются автомобили пяти типов: легковые, грузовые, автобусы, бензовозы, специальные машины.

Реализация модели выполнена с помощью внутренних средств языка моделирования. Использование дополнительного интерфейса для более удобной работы с моделью (изменение ключевых параметров системы, отображение автомобильного потока в динамике, отображение результатов на плане тоннеля) выполнено в виде внешней библиотеки (DLL), подключаемой к модели путепровода.

Результат моделирования отображен в таблице 1.

Таблица 1.

Результаты моделирования

Полоса	Результат по зонам (вероятность возникновения ЧС, %)										
	1	1-2	2	2-3	3	3-4	4	4-5	5	5-6	6
1	1,63	0,36	0,54	1,63	0,36	0,54	0,18	1,81	0,54	0,18	3,62
2	0,36	0,54	1,81	0,36	0,54	0,36	1,81	0,54	0,36	0,36	0,54
3	1,99	5,25	5,43	0,18	5,43	6,88	1,81	1,63	3,62		
4	5,07	1,81	3,44	3,44	0,54	5,43	0,18	0,18	0,36		
5	1,81	3,80	1,99	1,99	1,81	0,18	5,07				
6	1,99	0,36	0,18	0,54	3,44	1,63	3,44				

По результатам моделирования можно сделать вывод, что самым опасным участком путепровода является зона два-три на третьей полосе.

Предложенный подход и программное обеспечение иерархического метода исследования дискретных устройств позволяет эффективно выполнять анализ правильности функционирования устройств данного класса.

Список литературы:

1. Рудаков И.В., Смирнов А.А. Исследование сложных дискретных систем на базе агентного метода // Вестник МГТУ им. Н.Э.Баумана. Сер. Приборостроение.- 2009.- №3.- С.33-41.
2. Рудаков И.В., Давудпур М. Алгоритм декомпозиции формальной модели функционального блока дискретного устройства // Вестник МГТУ им. Н.Э.Баумана. Сер. Приборостроения.- 2006.- №1.- С.90-98.
3. Рудаков И.В., Давудпур М. Декомпозиционный метод исследования дискретных устройств // Информационные технологии.- 2006.- №2.- С.44-49.
4. Гнеденко Б.В., Коваленко И.Н. Введение в теорию массового обслуживания. М.: Изд-во ЛКИ. 2007. 400 с.
5. Лоу А.М., Кельтон В.Д. Имитационное моделирование. 3-е изд. - СПб: Питер, 2004. -847 с. - (Классика Computer Science). [Law A.M., Kelton W.D. Simulation Modelling and Analysis, McGraw-Hill, 2000.].
6. Карпов Ю. Имитационное моделирование систем. Введение в моделирование с Anylogic5. СПб.: «БХВ-Петербург». 2006. 400 с.
7. Павловский Ю.Н., Белотелов Н.В., Бродский Ю.И. Имитационное моделирование. М.: «Академия». 2008. 235 с.

Method of hierarchical study of complex discrete structures

06, June 2012

DOI: 10.7463/0612.0370230

Rudakov I.V.

Russia, Bauman Moscow State Technical University
irudakov@yandex.ru

The proposed approach to the study of complex hierarchical structures allows to carry out discrete analysis of a complex structure functioning with the use of macro-modeling methods and subsequent automatic decomposition. Practical implementation of hierarchical investigation of complex discrete structures is based on technology of object-oriented programming. A language oriented to transacts was developed as a tool of analysis and verification of structure simulation. The method was tested in simulation of traffic flow in an overpass tunnel type.

Publications with keywords: [modelling](#), [decomposition](#), [discrete structure of the complex](#), [programming model](#)

Publications with words: [modelling](#), [decomposition](#), [discrete structure of the complex](#), [programming model](#)

References

1. Rudakov I.V., Smirnov A.A. Issledovanie slozhnykh diskretnykh sistem na baze agentnogo metoda [The study of complex discrete systems on the basis of the agent method]. *Vestnik MGTU im. N.E. Baumana. Ser. Priborostroenie* [Herald of the Bauman MSTU. Ser. Instrumentation], 2009, no. 3, pp. 33-41.
2. Rudakov I.V., Davudpur M. Algoritm dekompozitsii formal'noi modeli funktsional'nogo bloka diskretnogo ustroistva [Algorithm for decomposition of a formal model of the functional block of the discrete devices]. *Vestnik MGTU im. N.E. Baumana. Ser. Priborostroenie* [Herald of the Bauman MSTU. Ser. Instrumentation], 2006, no. 1, pp. 90-98.
3. Rudakov I.V., Davudpur M. Dekompozitsionnyi metod issledovaniia diskretnykh ustroistv [Decomposition method for studying the discrete devices]. *Informatsionnye tekhnologii*, 2006, no. 2, pp.44-49.
4. Gnedenko B.V., Kovalenko I.N. Vvedenie v teoriyu massovogo obsluzhivaniia [Introduction to queuing theory]. Moscow, LKI Publ., 2007. 400 p.

5. Law A.M., Kelton W.D. *Simulation Modelling and Analysis*, McGraw-Hill, 2000. (Russ. ed.: Lou A.M., Kel'ton V.D. *Imitatsionnoe modelirovanie*. St. Petersburg, Piter, 2004. 847 p. (*Klassika Computer Science*)).
6. Karpov Iu. *Imitatsionnoe modelirovanie sistem. Vvedenie v modelirovanie s Anylogic5* [Simulation modeling of systems. Introduction to modeling with Anylogic5]. St. Petersburg, BKhV-Peterburg, 2006. 400 p.
7. Pavlovskii Iu.N., Belotelov N.V., Brodskii Iu.I. *Imitatsionnoe modelirovanie* [Simulation Modelling]. Moscow, Akademiia, 2008. 235 p.