Министерство образования Российской Федерации

САНКТ-ПЕТЕРБУРГСКИЙ
ГОСУДАРСТВЕННЫЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Ю.Б. Колесов

Объектно-ориентированное моделирование сложных динамических систем

Санкт-Петербург
Издательство СПбГПУ
2004

УДК 681.3

Колесов Ю.Б. Объектно-ориентированное моделирование сложных динамических систем. СПб.: Изд-во СПбГПУ, 2004. 240 с.

В монографии рассматривается проблема создания многокомпонентных гибридных моделей с использованием связей общего вида. Такие компьютерные модели необходимы, например, для объективной оценки принимаемых решений при объектно-ориентированном анализе сложных технических систем. Рассмотрены существующие подходы к решению этой проблемы и предложен оригинальный подход, основанный на использовании формализма гибридного автомата для численного моделирования. В работе предложено обобщение гибридного автомата, позволяющее построить достаточно мощный визуальный язык объектно-ориентированного моделирования и визуальный язык управления вычислительным экспериментом. Рассмотрены архитектура инструментальных программных средств компьютерного моделирования, поддерживающих эти языки и алгоритмы работы исполняющей системы. Основные положения предлагаемого подхода реализованы в конкретном пакете моделирования Model Vision, свободная версия которого доступна для всех желающих.

Предлагаемый вашему вниманию монография обобщает опыт многолетней работы автора. Монография рассчитана на научных работников, инженеров, аспирантов и студентов, интересующихся проблемами современного компьютерного моделирования.

Печатается по решению редакционно-издательского совета Санкт-Петербургского государственного политехнического университета.

Оглавление.

Введение.

Исторически сложилось так, что объектно-ориентированный подход в настоящее время в основном ассоциируется с объектно-ориентированным программированием (даже аббревиатура ООП обычно трактуется именно так), несмотря на то, что сам этот подход был впервые сформулирован в описании языка моделирования Simula-67 [19]. Однако, в последующие примерно пятнадцать лет объектно-ориентированный подход развивался почти исключительно в программировании (Smalltalk, C++, Object Pascal, Java). Своеобразным итогом тридцатилетнего развития объектно-ориентированного программирования можно считать появление «унифицированного языка моделирования» UML, предназначенного для создания объектно-ориентированных спецификаций программных систем на ранних этапах разработки [12]. И только к началу 1990-х гг. объектно-ориентированный подход стал проникать в область «традиционного» непрерывного и непрерывно-дискретного моделирования [86]. Появился термин «объектно-ориентированное моделирование» (ООМ). Одной из причин использования ООМ является то, что ООМ позволяет удобным образом решать ряд типовых задач моделирования, а именно:

- создавать библиотеки типовых компонентов как библиотеки классов;
- повторно использовать компоненты с помощью наследования классов;
- естественным образом строить модели с множеством однотипных объектов или регулярными структурами (массивами) объектов;
- осуществлять параметризацию моделей с помощью полиморфизма;
- при моделировании систем с переменным составом создавать и уничтожать экземпляры объектов в ходе вычислительного эксперимента.

Другой важной мотивацией использования ООМ является все более широкое использование объектно-ориентированного анализа при разработке сложных технических систем. Принятый настоящее время в качестве стандарта языка

объектно-ориентированного моделирования язык UML предлагает для фиксации результатов анализа ряд графических нотаций – диаграмм. Все эти диаграммы, отражающие различные аспекты разрабатываемой системы, являются (за исключением диаграммы состояний) неформальными и семантика выполняемых действий задается на уровне комментариев. В то же время для систем со сложной динамикой полноценный объектно-ориентированный анализ должен включать объективное тестирование сценариев желаемого поведения, а также параметрический синтез и оптимизацию. Для выполнения этих задач необходима компьютерная модель всей проектируемой системы в целом на высоком уровне абстракции. Обобщение свойств сложных технических систем как объекта моделирования приводит к понятию сложной динамической системы.

В данной работе предлагается подход к объектно-ориентированному моделированию сложных динамических систем, основанный на использовании формализма гибридного автомата.

В главе 1 рассматриваются вопросы объектно-ориентированного анализа при разработке сложных технических систем, вводится понятие сложной динамической системы и формулируются требования к инструментальным средствам автоматизации моделирования сложных динамических систем. В главе 2 проводится анализ существующих подходов к моделированию сложных динамических систем и существующих языков объектно-ориентированного моделирования. В главе 3 предлагается математическая модель обобщенного гибридного автомата как базовая для определения компонента модели, а также рассматриваются методы автоматического получения математической модели всей системы в целом. В главе 4 описываются принципы построения объектно-ориентированного языка компонентного моделирования сложных динамических систем на базе математической модели обобщенного гибридного автомата. В главе 5 рассматривается и обосновывается архитектура инструментальных программных средств автоматизации объектно-ориентированного моделирования сложных динамических систем.

В приложениях приведен ряд примеров, иллюстрирующих предлагаемый подход.

Все основные положения предлагаемого подхода реализованы в инструментальном пакете автоматизации моделирования Model Vision Studium. Свободно распространяемая версия этого пакета доступна через Интернет (например, на образовательном сайте www.exponenta.ru в разделе «Другие пакеты»).

Глава 1. Сложные динамические системы и их модели.

В данной главе рассматриваются вопросы объектно-ориентированного анализа при разработке сложных технических систем, вводится понятие сложной динамической системы и формулируются требования к инструментальным средствам автоматизации их моделирования.

Моделирование при объектно-ориентированном анализе сложных технических систем.

Во второй половине прошлого столетия в ряде областей техники (преимущественно военного направления) появились т.н. «сложные технические системы» или «технические комплексы» [4,44,49], к которым прежде всего относятся сложные системы управления динамическими объектами. Можно выделить следующие характерные особенности сложных технических систем [40,20,28]:

- элементы системы имеют разнородные физические принципы действия (электрические, механические, гидравлические, оптические и др. системы);
- между элементами системы, а также с внешней средой имеется множество связей, как информационных, так и физических:
- система имеет иерархическую многоуровневую структуру;
- имеется много различных режимов работы, причем эти режимы не совпадают, то есть, один режим работы одной подсистемы может требовать переключений режимов работы других подсистем;
- имеется значительная неопределенность в поведении объектов управления и внешней среды;
- устройства управления помимо задач обычного непрерывного управления решают также задачи логического управления, диагностики и др.;
- большая часть функций управления реализуется программно на встроенных ЭВМ и микропроцессорах;

- очень часто программное обеспечение и аппаратура разрабатываются одновременно;
- часто состав и структура системы изменяется в ходе ее функционирования.

К «традиционным» сложным техническим системам относятся ракетные и космические комплексы, комплексы противовоздушной и противоракетной обороны, некоторые АСУ ТП и др. В последнее десятилетие роль сложных технических систем резко возросла. Благодаря прогрессу микроэлектроники появились дешевые, надежные и быстродействующие встроенные микропроцессоры и ЭВМ. Это привело, во-первых, к усложнению алгоритмов управления и контроля в «традиционных» сложных системах, а во-вторых, к появлению программной реализации функций управления и контроля во все большем числе технических объектов. Некоторые характерные черты сложных систем появились даже в таких «бытовых» технических системах как автомобиль, стиральная машина, микроволновая печь и т.п. [17]. Соответственно расширился и круг инженеров-проектировщиков, занятых разработкой и сопровождением сложных технических систем.

В современных сложных технических системах значительная доля трудоемкости разработки приходится на разработку программного обеспечения (ПО) встроенных ЭВМ и микропроцессоров [17,55,42]. Многочисленные ошибки в этом ПО приводят к затягиванию этапов динамической комплексной отладки и испытаний, а также к неожиданным отказам системы во время эксплуатации. Эти ошибки обусловлены прежде всего логической сложностью комплекса программ, не случайно число изменений в программных модулях, координирующих работу подсистем, на порядок превышает число изменений в модулях, реализующих отдельные функции [41,42]. С 1970-х гг. активно разрабатываются методологии структурного проектирования сложных программных комплексов, такие как SADT [104], IDEF [61], метод Йордана [118] и др. С конца 80-х годов начали также интенсивно развиваться объектно-ориентированные методологии разработки программного обеспе-

чения. В настоящее время объектно-ориентированный подход считается наиболее современным и прогрессивным [17,11,55]. В 1997 г. OMG (Object Management Group) приняла язык UML [78,12], появившийся в результате слияния ряда известных методологий, в качестве стандарта языка объектно-ориентированного моделирования (в данном случае это скорее «прототипирование») при разработке ПО. В настоящее время уже существуют разработанные рядом компаний CASE-средства, поддерживающие язык UML, например, такой известный продукт как Rational Rose [7]. Существуют также и отличные от UML объектно-ориентированные методологии, например методология ROOM для разработки систем реального времени [116], а также различные комбинации структурного и объектного подходов [55]. В данной работе мы будем ориентироваться на понятийный аппарат языка UML. Практикой показано, что при разработке сложного программного обеспечения самые принципиальные просчеты делаются на самых ранних этапах разработки и что обнаружение и устранение этих ошибок на ранних этапах в десятки и сотни раз быстрее и дешевле, чем на завершающих этапах разработки и испытаний [42]. Поэтому в объектно-ориентированной разработке сложных технических систем особенно важен этап объектно-ориентированного анализа. Последний определяет так много важных требований к инструментальным средствам ООМ, что мы рассмотрим его подробней.

Объектно-ориентированный анализ в основном должен выполняться на стадиях НИР и ОКР. В идеальном случае анализ должен выполняться один раз (так называемая «водопадная» модель процесса проектирования), однако, все же более жизненной является «итеративная» модель, допускающая возврат к анализу системы на последующих этапах разработки с целью уточнения спецификаций системы [17,101]. Целью объектно-ориентированного анализа является [17,11]:

- выявление объектов и их связей, то есть функциональной структуры системы;

- определение желаемого поведения системы в основных режимах работы, так называемых «сценариев», в возможно более формальном виде;
- выделение классов объектов и отношений между классами;
- определение границы между аппаратной и программной составляющими системы.

Совокупность результатов объектно-ориентированного анализа в работе [17] называется «аналитической моделью» проектируемой системы, а процесс ее создания - «аналитическим моделированием». Собственно, аналитическая модель и есть единственная модель, которую имеет смысл при разработке сложной СУ, поскольку модели более низких уровней абстракции сравнимы по трудоемкости создания с разработкой самой системы [49,40,5]. Поскольку слово «аналитический» часто понимается как синоним «символьный», мы будем далее использовать термин «системно-аналитический».

Объектом принято называть некоторую сущность, которая инкапсулирует в себе данные (атрибуты объекта) и поведение как единое целое и взаимодействует с внешним окружением через определенный интерфейс [19,78,107,77,97]. Объекты функционируют параллельно и независимо от других объектов. Каждый объект является экземпляром некоторого класса. Практически невозможно указать единый принцип выделения объектов, пригодный для любой системы. Обычно при выделении объектов руководствуются либо физической структурой системы, либо функциональными ролями элементов системы [17,11]. Типичным можно считать случай, когда в начале разработки сложной технической системы имеется некоторый набор уже готовых аппаратных блоков (типовых или переносимых из предыдущих разработок с небольшими модификациями), некоторый набор аппаратных блоков разрабатывается вместе с системой, а архитектуру управляющего цифрового вычислительного комплекса (УЦВК) еще предстоит разработать (Рис. 1).

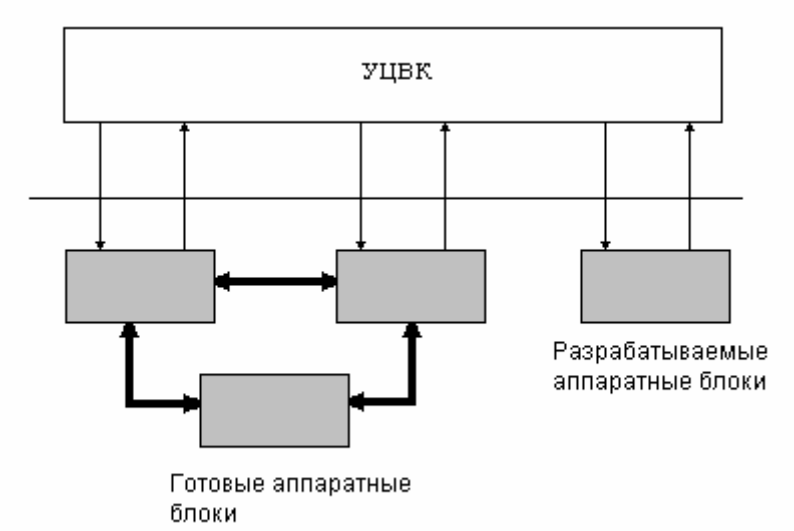


Рис. 1

На Рис. 1 заданные физические подсистемы («приборы») показаны закрашенными прямоугольниками, толстыми линиями показаны физические связи между ними, тонкими линиями показаны информационные связи. Заметим, что «физические» подсистемы могут сами содержать свои внутренние встроенные ЭВМ и/или микропроцессоры с «прошитым» собственным ПО. Однако, если это ПО не является предметом разработки, то вся подсистема рассматривается как аппаратная. Таким образом, на самых ранних этапах проектирования сложной системы как правило существует значительная неопределенность в физической структуре системы и конкретных аппаратных решениях. Так называемое «сопроектирование» аппаратной и программной составляющих («HW/SW Codesign») является характерной особенностью процесса разработки значительной части современных сложных систем [75,95]. Окончательная граница раздела «программы - аппаратура» может сама являться одним из результатов анализа и неоднократно уточняться позднее [76]. Поэтому представляется более предпочтительным при выделении объектов системы руководствоваться их функциональными ролями.

Методология объектно-ориентированного проектирования рекомендует на стадии объектно-ориентированного анализа выделять только активные объекты, функционирующие независимо и параллельно [17]. Применительно

к сложным техническим системам наиболее естественным представляется рассматривать в качестве основного при выделении активных объектов отношение «объект управления (ОУ) – устройство управления (УУ)». Аппаратные объекты управления («внешние» объекты в терминологии UML [17,12]) являются активными по определению. Активные программные объекты являются устройствами управления соответствующего уровня. Совокупность функционирующих независимо и параллельно взаимодействующих активных объектов отражает функциональную структуру проектируемой системы (Рис. 2).

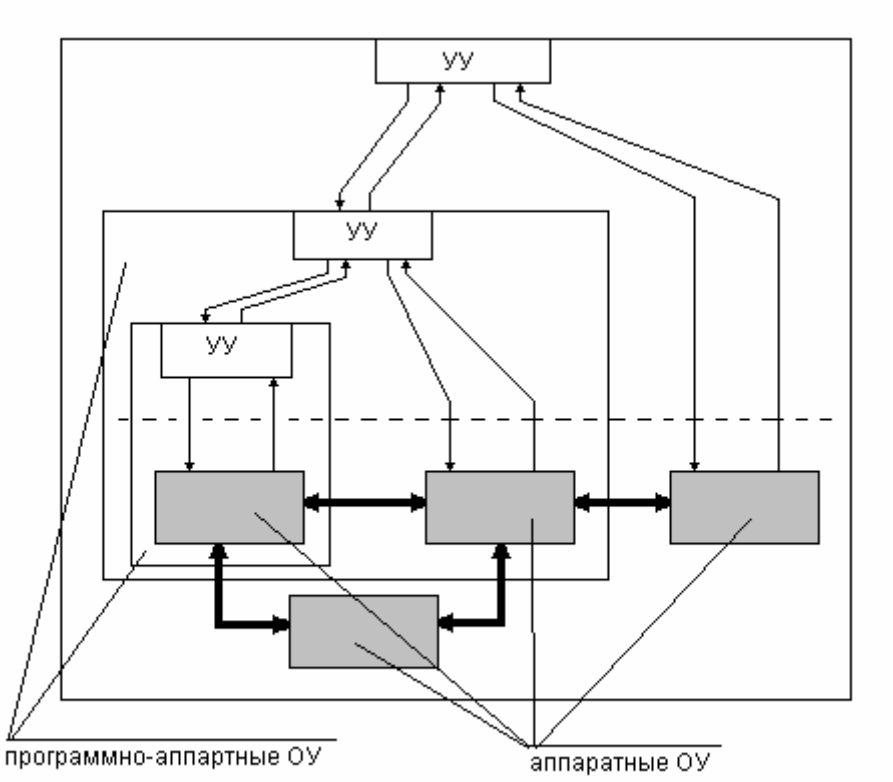


Рис. 2

При выделении объектов «по управлению» часть составных объектов становится программно-аппаратными. Программно «надстраивая» аппаратуру, мы получаем более высокоуровневую и «интеллектуальную» подсистему. Такие подсистемы соответствуют тому, что в работе [40] называется «виртуальным устройством», а в работе [45] «`control configured vehicle`». Вообще все внешние объекты на этапе объектно-ориентированного анализа

можно рассматривать как программно-аппаратные, поскольку точный интерфейс взаимодействия с реальным физическим объектом может быть на этом этапе просто неизвестен. На последующих этапах проектирования можно уточнить этот внешний объект, используя отношение наследования, и выделить его программную составляющую. Иногда физическая подсистема по разным соображениям интегрирует в себе несколько функциональных аспектов. В этом случае в качестве локального объекта управления в функциональной подсистеме целесообразно использовать соответствующий интерфейс физического объекта (Рис. 3).

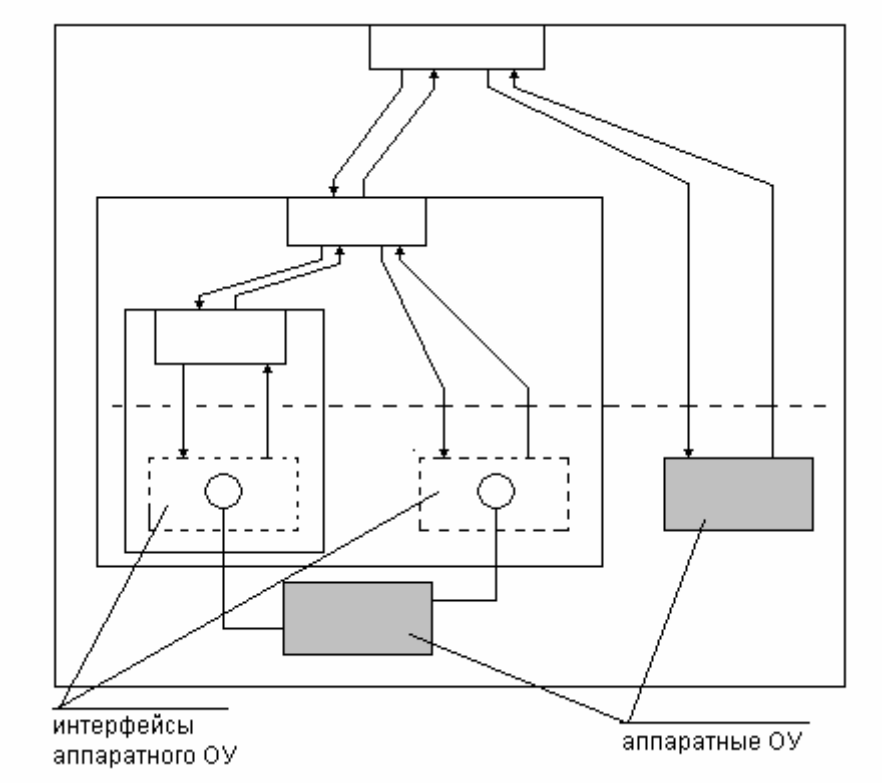


Рис. 3

«Традиционные» задачи непрерывного управления (регулирования) обычно характерны для нижних уровней иерархии сложных систем. На верхних уровнях иерархии обычно имеют место процессы так называемого логического управления [31,8,67]. Например, УУ верхнего уровня автоматического сверлильного станка осуществляет координацию согласованной работы подсистем управления нижнего уровня: включает вращение шпинделя, включает вертикальную подачу, по контакту сверла с деталью включает по-

дачу охлаждающе жидкости и отключает ее при исчезновении контакта, при достижении заданной глубины отверстия меняет направление вертикальной подачи, контролирует поломку сверла и т.д. В то же время УУ нижних уровней регулируют значения отдельных показателей, например, поддерживают заданную скорость вращения шпинделя.

В руководствах по объектно-ориентрованному анализу [17,11] желаемое поведение рекомендуется определять в виде сценариев для основных режимов работы системы. В первую очередь рекомендуется составлять «первичные» сценарии, отражающие нормальную работу системы. Затем должны быть разработаны «вторичные» сценарии, отражающие реакцию системы на различные исключительные ситуации (отказ аппаратуры, неверные исходные данные, отмена режима оператором и т.п.). Для задания сценариев могут быть использованы две конструкции UML: диаграмма взаимодействий и диаграмма состояний [12].

Один из видов диаграммы взаимодействий – диаграмма последовательностей – прекрасно знакома большинству разработчиков алгоритмов функционирования сложных систем как «циклограмма режима». Диаграмма последовательностей предназначена для фиксации временной последовательности событий в системе. В верхней части диаграммы по оси X указываются объекты, участвующие во взаимодействии, причем инициирующие взаимодействие объекты располагаются левее, а подчиненные правее. Применительно к СУ это соответствует иерархии уровней управления. По оси Y размещаются пунктирные «линии жизни» объектов во времени и показывается стрелками передача сообщений от одного объекта к другому (более поздние показываются ниже). Кроме того, для программных объектов на «линиях жизни» вытянутыми прямоугольниками может быть показан «фокус управления». Применительно к активным объектам «фокус управления» не особенно актуален, поэтому можно показывать условные переходные процессы, вызванные поступившими сообщениями. Диаграммы последовательностей особенно наглядны для процессов логического управления. Со-

общения, поступающие «слева направо» от контроллера к объекту управления ассоциируются с командами, а сообщения, поступающие «справа налево» с донесениями. Другим видом диаграмм взаимодействий являются диаграммы кооперации, на которых изображаются объекты, участвующие во взаимодействии и их связи с указанием передаваемых сообщений. Для СУ это по существу фрагмент структурной схемы. Диаграммы взаимодействий безусловно важны для самой предварительной проработки сценариев работы системы. Вообще говоря, само выделение набора программно реализуемых объектов и их связей практически возможно только после нескольких итераций построения сценариев. Однако эти диаграммы являются неформальными, семантика выполняемых действий задается на уровне комментариев.

Для однозначного понимания, возможности объективного тестирования желаемого поведения, а также для последующего использования в качестве имитаторов на стадии проектирования необходима формальная математическая модель, отражающая структуру и поведение разрабатываемой системы на самом высоком уровне абстракции. Кроме того, в процессе анализа и сравнения вариантов могут потребоваться модели отдельных подсистем, причем различных уровней детализации. Анализ предполагает оценку правильности принимаемых решений. Для сложных систем такую оценку можно получить только путем проведения вычислительных экспериментов с компьютерной моделью системы по разработанным сценариям поведения. На этапе объектно-ориентированного анализа должны решаться также задачи синтеза и оптимизации. Для решения этих задач также необходимы математические и компьютерные модели разрабатываемой системы.

Единственным формальной конструкцией UML является диаграмма состояний, которая является чуть измененной картой состояний Харела [93,94]. Карта состояний представляет собой направленный граф, вершины которого связываются с качественными состояниями системы, а дуги с переходами из одного состояния в другое. Переход может произойти по истечении некоторого интервала времени, по выполнении логического условия и по

дискретному событию (сообщению, исключительной ситуации и т.п.). Карта состояний является чрезвычайно удобным и наглядным инструментом для описания взаимодействующих дискретных параллельных процессов, развивающихся в непрерывном времени. Функционирование программной составляющей системы можно описать с помощью карт состояния достаточно адекватно. Иногда с помощью карты состояний можно достаточно адекватно описать и поведение аппаратных объектов управления. Например, в [55] показано успешное использование несколько модифицированной карты состояний для описания системы управления телефонной АТС. Однако, карты состояний недостаточно для описания поведения непрерывных объектов управления. Возможна, конечно, «дискретная аппроксимация» непрерывного поведения объектов управления, когда оно заменяется в упрощенных моделях на совокупность чисто дискретных переходных процессов, являющихся реакцией на определенные управляющие воздействия. Например, непрерывное поведение гироплатформы в режиме грубого горизонтирования можно заменить карту состояний, показанную на Рис. 4.

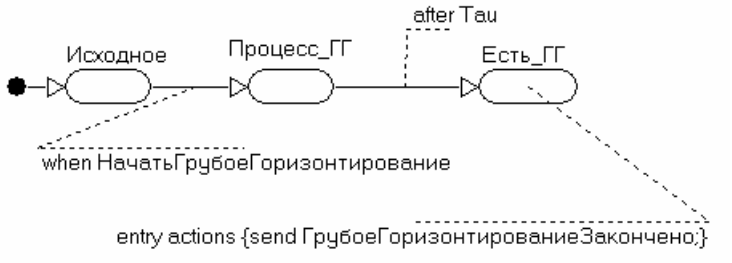


Рис. 4

В принципе такой подход, конечно, допустим, но имеет ряд серьезных недостатков. Полученную в результате «дискретной аппроксимации» модель очень трудно модифицировать в случае, когда потребуется более детальное рассмотрение динамики процесса (например, для исследования перекрестного влияния подсистем). При дискретной «аппроксимации» достаточно сложно имитировать ошибочные ситуации для отработки «вторичных» сценариев поведения. На взгляд автора правильнее непрерывные объекты всегда пред-

ставлять непрерывными моделями, пусть даже чрезвычайно упрощенными (например, полагать систему линейной).

Типовая сложная СУ в целом является гибридной системой. Под гибридными системами понимаются системы, демонстрирующие как непрерывные, так и дискретные аспекты поведения [4,34,110].. В литературе также используются термины «непрерывно-дискретные системы», «системы с переменной структурой», «событийно-управляемые» [10,27]. Обычно выделяют три основных фактора, приводящих к необходимости использования гибридной модели:

1) совместное функционирование непрерывных и дискретных объектов;

2) качественные изменения в непрерывном объекте;

3) изменение числа и состава непрерывных объектов, входящих в систему, в ходе функционирования.

Все эти три фактора имеют место среди основных особенностей сложной СУ.

Таким образом, для полноценного выполнения объектно-ориентированного анализа при разработке сложной СУ необходима компьютерная модель всей СУ в целом. Создание такой модели вручную практически невозможно из-за огромных трудозатрат и ненадежности результатов экспериментов с такой моделью. Опыт показывает, что программирование компьютерных моделей вручную возможно только для относительно стабильных подсистем и отработка таких моделей занимает годы. Следовательно, объектно-ориентированный анализ сложных СУ невозможен без средств автоматизации моделирования.

Таким образом, обобщая свойства сложных технических систем как объекта моделирования, мы приходим к понятию сложной динамической системы. Сложной динамической системой (СДС) будем называть систему, обладающую следующими свойствами:

- система состоит из многих компонентов, состав которых может изменяться во время функционирования системы;
- компоненты имеют различную физическую природу;

- между компонентами имеются как ориентированные, так и неориентированные связи;
- система может иметь иерархическую многоуровневую структуру;
- элементарные компоненты могут быть непрерывными, дискретными или гибридными.

Следует отметить, что к СДС могут сводиться, конечно, и другие системы помимо технических.

Требования к инструментальным средствам моделирования.

Прежде всего, уточним, какого типа инструментальные средства необходимы для моделирования СДС на этапе объектно-ориентированного анализа. Инструментальные средства (пакеты) для моделирования можно разделить на две группы (Рис. 5) [35].

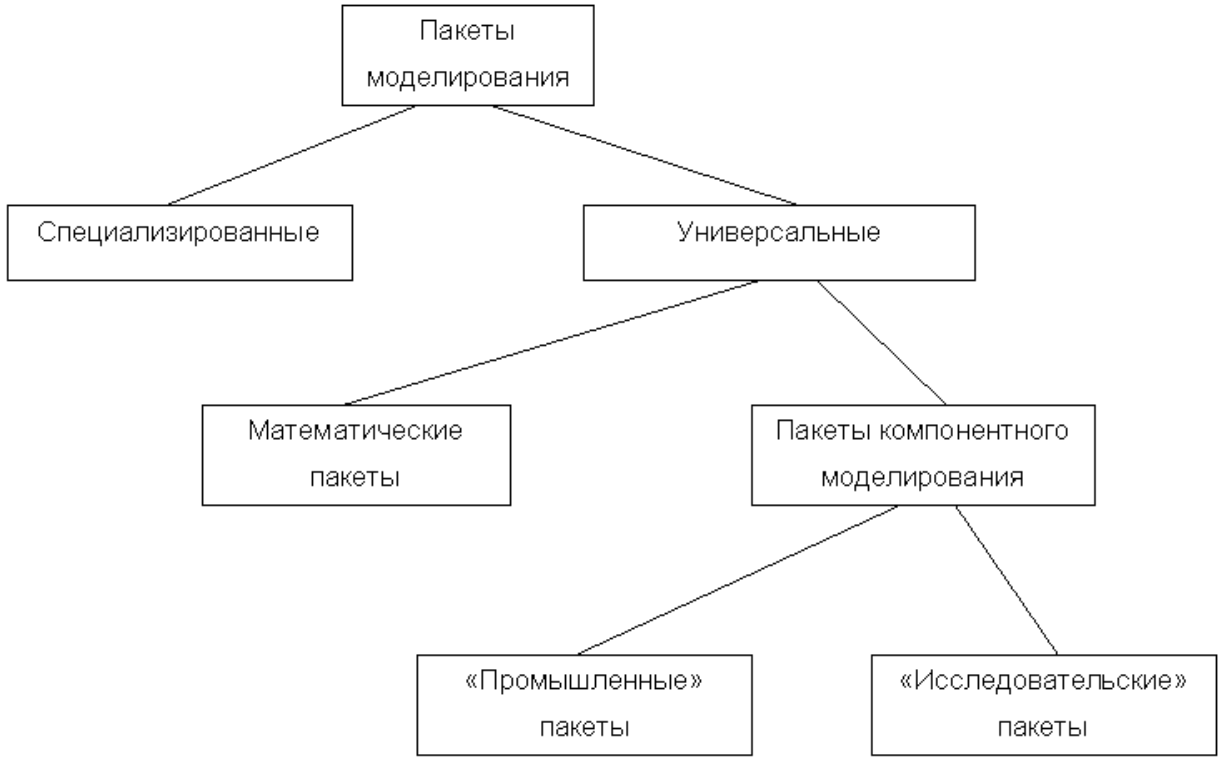


Рис. 5

К первой относятся специализированные программные средства (пакеты), ориентированные на специфические понятия конкретной прикладной

области (химической технологии, теплотехники, электротехники и т.д.). Ко второй относятся так называемые «универсальные» пакеты, ориентированные на определенный класс математических моделей и применимые для любой прикладной области, в которой эти модели пригодны. Ясно, что специализированные пакеты могут быть использованы только для автономного исследования отдельных подсистем СДС. Полученные специализированные модели чрезвычайно сложно использовать в дальнейшем для построения комплексной модели СДС, а также для отработки программ на этапе проектирования. Поэтому даже при моделировании отдельных подсистем предпочтительно использовать «универсальные» пакеты.

«Универсальные» пакеты далее обычно разделяют на «математические» пакеты и пакеты компонентного моделирования (Рис. 5). В «математических» пакетах (Mathematica, MathCAD, MatLab, Maple [52]) предполагается, что математическая модель всей моделируемой системы уже каким-либо образов построена и ее требуется исследовать. Такой подход характерен в основном для научных исследований. Как правило, математические пакеты сочетают численные эксперименты с символьными преобразованиями. Компонентное моделирование предполагает, что описание моделируемой системы строится из компонентов (в том числе и готовых библиотечных), а совокупная математическая модель формируется пакетом автоматически. Пакеты компонентного моделирования в основном ориентированы на численные эксперименты. Компонентное моделирование преобладает в процессе проектирования технических объектов. Очевидно, что для СДС построение и сопровождение ее полной математической модели вручную практически невозможно. Следовательно, для аналитического моделирования должны использоваться пакеты компонентного моделирования, которые по способам их применения или технологии моделирования также можно разделить на две группы.

К первой отнесем пакеты, предназначенные для решения сложных промышленных и научно-исследовательских задач большими производственными или научными коллективами. В таких проектах ведущую роль игра-

ет организация работ: хорошо налаженное взаимодействие между отдельными группами, быстрый доступ к многочисленным экспериментальным данным и библиотекам программ, тщательное документирование и тестирование, многовариантные расчеты. При этом обычно используются хорошо изученные готовые математические модели, которые лишь модифицируются и приспосабливаются для решения конкретных задач. Пользователи пакета подразделяются на две категории: разработчики библиотек готовых моделей и обычные пользователи, работа которых сводится к составлению схем из типовых блоков и параметрическая настройка блоков. Пакеты первой группы условно назовем «промышленными».

Совсем другая технология характерна для предварительных исследований, выполняемых отдельными учеными или проектировщиками. Библиотеки готовых моделей используются весьма ограничено. Исходным материалом служат плохо формализованные «сырые» модели, то есть модели, чьи свойства еще не вполне осознаны. Это означает, что необходимо уметь организовывать и поддерживать непрерывную обратную связь между исследователем и исследуемой моделью. Несмотря на большие достижения в области автоматического синтеза систем с заданными показателями, на практике разработка новой технической системы – это прежде всего просмотр большого числа пробных вариантов. Назовем пакеты второй группы «исследовательскими», подчеркивая этим, что они уступают по количеству уникальных возможностей промышленным, зато более просты для освоения и доступны отдельному исследователю при решении относительно несложных задач из практически любой прикладной области. Под «несложными» будем понимать не простые задачи, а задачи посильные одному разработчику, не являющемуся специалистом в области программирования и вычислений.

С «исследовательскими» пакетами тесно связана концепция активного вычислительного эксперимента [36], предусматривающая:

- визуализацию результатов моделирования не после эксперимента, а во время эксперимента;

- возможность интерактивного вмешательства пользователя в ход вычислительного эксперимента;
- возможность использования 2D и 3D-анимации, в том числе интерактивной.

Активный вычислительный эксперимент позволяет максимально быстро оценивать моделируемый вариант системы, имитировать различные отказы и т.д. Например, наличие интерактивной анимации в модели системы управления лифтами, рассматриваемой в [17], сразу позволило бы после недолгой «игры» с кнопками управления вывить множество логических ошибок в первых версиях модели. Из сказанного ясно, что инструмент системно-аналитического моделирования ССУ в основном можно отнести к «исследовательским» пакетам.

Имеется еще один аспект применения инструментальных средств системно-аналитического моделирования – техническое образование. Проектирование современных сложных технических систем требует помимо фундаментального и технического образования еще и некоторой минимальной инженерной практики, которая традиционно приобретается в первые годы работы молодого специалиста на предприятии. Предприятие по существу получает «полуфабрикат» инженера-проектировщика и «доводит» его до профессионального уровня. Кроме того, опыт показывает, что далеко не все инженеры склонны и способны к комплексному проектированию сложных систем. Частично такая практика может приобретаться уже в ВУЗ’е, если студентам предоставляется возможность в ходе учебного проектирования создавать на персональном компьютере визуальные макеты сложной системы. Возможность сразу наблюдать поведение созданной системы замыкает обратную связь в учебном процессе и учит находить инженерные компромиссы. Таким образом, появляется возможность выявлять студентов, склонных к работе проектировщика.

На основании анализа свойств СДС и особенностей процесса их проектирования можно сформулировать следующие требования к инструментальным средствам моделирования.

1. Требования к входному языку**:**

- входной язык должен быть объектно-ориентированным;
- входной язык должен поддерживать гибридные модели, возникающие при моделировании сложной технической системы на этапе объектно-ориентированного анализа;
- для описания чисто дискретных объектов должна использоваться карта состояний в нотации UML;
- для описания чисто непрерывных объектов должны использоваться системы уравнений в общепринятой математической нотации;
- входной язык должен поддерживать компонентное моделирование с использование как направленных, так и ненаправленных связей между компонентами
- входной язык не должен навязывать пользователю никаких дополнительных сущностей, обусловленных не моделируемой системой, а используемой формальной схемой;
- определение новых классов объектов и формирование библиотек классов должно выполняться средствами самого входного языка;
- входной язык должен включать в себя средства управления вычислительным экспериментом.

2. Требования к проведению вычислительного эксперимента**:**

- инструментальные средства должны быть «устойчивы» по отношению к математически некорректным моделям и выдавать пользователю внятную диагностику и рекомендации;
- инструментальные средства должны поддерживать концепцию активного вычислительного эксперимента;
- должна быть предусмотрена возможность работы модели в реальном времени;

- должна быть предусмотрена возможность внешнего управления моделью из другого программного приложения.

3. Требования к интегрированной среде**:**

- должен использоваться инкрементный транслятор, который контролирует правильность отдельной языковой конструкции немедленно по завершении ее ввода;
- должна обеспечиваться возможность импорта и экспорта данных;
- среда должна обеспечивать возможность подключения дополнительных приложений для выполнения специальных функций, например, параметрической оптимизации, частотного анализа и т.п.;
- выполняемая модель должна создаваться максимально быстро.

Анализ ряда известных современных программных средств автоматизации моделирования динамических систем (зарубежных, таких как MatLab [18], VisSim [84], MSC.Easy5 (www.mscsoftware.com), Dymola [82], а также отечественных, таких как МВТУ [30], Stratum [47,3,46], ИСМА [64]) показывает, что ни один из этих инструментов в полной мере не отвечает этим требованиям.

Глава 2. Анализ существующих подходов к объектно-ориентированному моделированию сложных динамических систем.

В данной главе проводится системный анализ существующих подходов к компонентному моделированию и моделированию гибридных систем, а также анализ существующих языков объектно-ориентированного моделирования.

Существующие подходы к компонентному моделированию.

В современных инструментах компонентного моделирования непрерывно-дискретных систем можно выделить два основных направления:

- направление «блочного моделирования»;
- направление «физического моделирования».

Инструменты т.н. «блочного моделирования» ориентированы на графический язык иерархических блок схем. Элементарные блоки являются либо предопределенными, либо могут конструироваться с помощью некоторого специального вспомогательного языка более низкого уровня. Собранную схему можно объявить типовым блоком следующего уровня – подсистемой. Схему можно собрать из имеющихся блоков с использованием направленных связей и параметрической настройки. Для некоторых специальных наборов блоков (например, механических [2] или электрических [56]) возможно использование и ненаправленных связей. Несмотря на то, что уже два десятилетия назад этот подход объявлялся пережитком аналогового моделирования [50], он процветает и не только не собирается вымирать, но и доминирует в настоящее время. Появление персональных компьютеров и графического дисплея вдохнуло в него новую живительную струю. Наиболее известными современными представителями этого направления являются: подсистема

SIMULINK пакета MATLAB (MathWorks, Inc.), EASY5 (MSC Software); VisSim (Visual Solution); МВТУ (МГТУ им. Н.Э.Баумана). В качестве типичного представителя этого направления мы будем далее рассматривать SIMULINK [4,18,62]. Очевидно, что этот подход не может быть использован для компонентного моделирования СДС, так как, во-первых, принципиально не поддерживает мультидоменные компонентные модели с ненаправленными связями и , во-вторых, не позволяет задавать описание элементарных компонентов средствами входного языка, что приводит к появлению в модели искусственных структурных схем, не соответствующих структуре моделируемой системы.

Направление т.н. «физического моделирования» представлено пакетами Dymola [82] и MathModelica [90], поддерживающими новый унифицированный объектно-ориентированный язык моделирования физических систем Modelica [107,108]. Этот новый язык объединил результаты, достигнутые в области моделирования электрических, механических, гидравлических и др. систем. Его отличительными особенностями являются отказ от ориентированных связей между компонентами, а также возможность пользователю самому задавать описание новых компонентов, определяя их поведение в виде системы дифференциально-алгебраических уравнений, задаваемых в достаточно свободной форме. Вообще для этого направления характерно стремление свести все явления к одной большой системе уравнений, его лозунгом могла бы быть фраза «Все есть уравнения!». При таком подходе совокупная система уравнений всей модели в целом с учетом уравнений связей может сильно отличаться от простого объединения уравнений компонентов. Внешние переменные компонентов, участвующие в ненаправленных связях, подразделяются на «контакты» и «потоки». Соединению «контактов» соответствует равенство всех соединяемых переменных, соединению «потоков» соответствует равенство суммы всех соединяемых переменных нулю. Подход Modelica в части компонентного мультидоменного моделирования полно-

стью отвечает всем требованиям, предъявляемым к компонентному моделированию СДС.

Существующие подходы к моделированию гибридных систем.

Гибридной называется система, демонстрирующая как непрерывные, так и дискретные аспекты поведения [4,34,110].. В литературе также используются термины «непрерывно-дискретные системы», «системы с переменной структурой», «реактивные», «событийно-управляемые» [10,27]. В данной главе под непрерывным будем понимать объект, состояние которого определяется вектором $x \in \Re^n$, а поведение задается любым отображением $C : \Re^n \to \Re^n$, определяющее функцию $x = x(t; x^0)$, для которой выполняются следующие утверждения:

- функция непрерывна по совокупности переменных;
- $x(0; x^0) = x^0$;
- $x(t_2; x(t_1; x^0)) = x(t_1 + t_2; x^0)$.

где $\Re$ - множество вещественных чисел, $t \in \Re$ - непрерывное время. Конкретные формы отображения C будут рассмотрены в Главе 2. Под дискретным в данной главе будем понимать объект, состояние которого определяется вектором $d \in W^n | W = \Re \cup Z \cup B$ (где Z - множество целых чисел, $B = \{false, true\}$ - множество булевских величин), а поведение задается отображением $D : N \to \Re \times W^n$, которое определяет причинно-следственную цепочку дискретных событий $(0, d_0) \to (t_1, d_1) \to (t_2, d_2) \to ... \to (t_i, d_i)$.

Гибридное поведение в модели сложной динамической системы.

Ранее были отмечены три основных фактора, обуславливающих появление гибридного поведения. Рассмотрим их более подробно.

Гибридное поведение, обусловленное совместным функционированием непрерывных и дискретных объектов.

Такое гибридное поведение характерно для систем автоматического управления, в которых имеется непрерывный объект управления и дискретное устройство управления (контроллер). Обычно предполагается, что на интервале $[t_i, t_{i+1})$ $d = d_i$, то есть зависимость $d(t, d_0)$ является кусочно-постоянной функцией (Рис. 6).

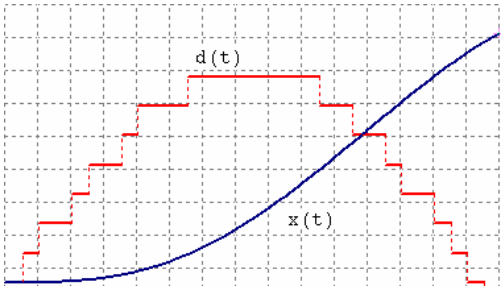


Рис. 6

В частном случае отображение D контроллера является периодическим с периодом T и задается разностными уравнениями. Для линейного случая используется дискретное преобразование Лапласа (z-преобразование). Именно такие системы обычно рассматриваются как дискретные системы автоматического регулирования [53] (Рис. 7).

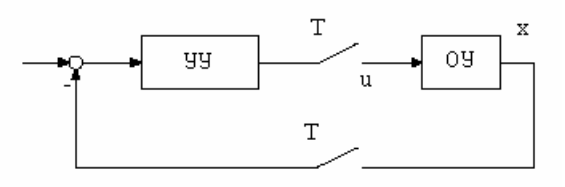


Рис. 7

Для верхних уровней управления в сложных иерархических системах управления характерны процессы логического управления [31,8,67]. В этом случае отображение D устройства управления является асинхронным процессом, в котором следующее дискретное событие (t_{i+1}, d_{i+1}) зависит в общем случае от предыдущего (t_i, d_i), а также от функции $x(t)$. Например, устройство логического управления ракетой выдает команду на отсечку тяги при дос-

тижении некоторым функционалом порогового значения. Для задания функций устройств логического управления использовались таблицы решений, конечные автоматы, сети Петри и другие формальные модели для описания асинхронных процессов [8]. В последнее время для этой цели все чаще используются карты состояния (statechart), предложенные Д.Харелом [93] и «канонизированные» в качестве стандарта de-facto для описания дискретных процессов в языке UML[78,12]. С помощью карт состояний можно также удобно и наглядно описывать и поведение периодических контроллеров [4].

Гибридное поведение, обусловленное мгновенными качественными изменениями в непрерывном объекте.

Некоторые непрерывные по своей природе системы могут демонстрировать дискретные черты поведения, связанные с происходящими в них качественными изменениями. Сами качественные изменения обусловлены прежде всего многорежимностью функционирования системы. Например, даже простейшая непрерывная система – бассейн с двумя трубами из школьного учебника -, рассматриваемая в чуть более широком контексте, немедленно демонстрирует качественные изменения – переполнение или опустошение бассейна. Следует отметить, что в отличие от первого и третьего типа гибридного поведения, где гибридное поведение является естественным изначальным свойством самой проблемы, гибридное поведение данного типа в определенной степени искусственно и связано исключительно с удобной для исследователя формализацией явления. В самом деле, поведение реальных физических систем всегда непрерывно. Дискретность появляется вследствие идеализации этого поведения. В работе [111] отмечается, что основными идеализациями такого рода являются пренебрежение временем переходных процессов, когда это время на порядки меньше времени функционирования исследуемой системы, а также идеализация параметрических зависимостей. Примером идеализации первого вида является пренебрежение временем абсолютно упругого отскока мячика от абсолютно твердой плоскости, а приме-

ром идеализации второго вида – идеализация реальной вольт-амперной характеристики для идеального диода (см. примеры в Приложении 1). Кроме того, исследователь может просто абстрагироваться от детального описания динамики нелинейного переходного процесса (или детальное описание может быть просто неизвестно), заменяя его некоторыми интегральными зависимостями. В результате такого рода абстракций в непрерывной системе в определенных временных точках t^* начинают происходить некоторые мгновенные дискретные изменения, то есть произойдет дискретное событие E^*. Следует отметить, что из-за своего искусственного характера гибридные модели этого вида могут демонстрировать парадоксальные поведения, которые не свойственны исходным непрерывным объектам и исследователь должен быть чрезвычайно осторожен при формализации гибридной модели.

Следует отметить, что в теории динамических систем и теории управления модели, описываемые различными уравнениями в различных областях фазового пространства известны очень давно. В работах [1,9,58] формулируются основные проблемы и делается попытка построить методы исследования гибридных систем. Дальнейшее развитие теории гибридных систем связано с решением задач стабилизации и слежения с помощью регуляторов, структура которых зависит от времени [22,54,57,23]. Однако, все эти результаты непосредственно не касаются численного моделирования.

Гибридное поведение, обусловленное изменением состава системы.

Если непрерывные объекты в ходе функционирования могут появляться в границах изучаемой системы и покидать ее, то состав совокупного вектора состояний всей системы x и его размерность будут изменяться. Примерами таких систем могут служить аэропорт (самолеты появляются в границах зоны аэропорта извне, садятся в аэропорту, взлетают из аэропорта), комплекс противовоздушной обороны (цели появляются в зоне обнаружения, выходят из нее, уничтожаются ракетами), система возникающих и исчезающих заряженных частиц и т.п. Модели такого типа с чисто дискретным

поведением элементов системы являются традиционными для т.н. имитационного моделирования [63]. Модели с простым непрерывным поведением (обычно это дифференциальные уравнения в форме Коши и формулы) поддерживаются рядом инструментов [50,16,92].

Типы гибридного поведения.

Очевидно, что все три рассмотренных выше случая появления гибридного поведения после перехода к рассмотрению совокупного поведения всей системы в целом сводятся к появлению дискретных событий в кусочно-непрерывном объекте. Результатом дискретного события в непрерывном объекте может являться:

1) скачкообразное изменение значения некоторых переменных $\{x_i \mid i \in J \subset 1..n\}$, то есть $x(t^* - 0) \neq x(t^* + 0)$. Примером такого дискретного события является упругий отскок мячика от плоскости, в результате которого вертикальная составляющая скорости мгновенно изменяет знак на противоположный (см. Пример 1 в Приложении 1). Такое скачкообразное изменение значений можно рассматривать как результат некоторой последовательности операторов присваивания $A^* = \{x_i := x_j^* \mid i \in J \subset 1..n\}$, выполняемой мгновенно в момент t^*.
2) изменение отображения C. Примером такого дискретного события является падение мячика на вертикально стоящую пружину, после которого в уравнении движения мячика появляется сила реакции пружины (см. Пример 2 в Приложении 1).
3) изменяется набор переменных $x = \{x_1, x_2, ..., x_n\}$ и возможно размерность n, причем они известны заранее. Примером такого дискретного события является обрыв маятника (см. Пример 3 в Приложении 1). После обрыва число степеней свободы маятника изменяется с 1 до 2 и изменяются уравнения движения;

4) изменяется набор переменных $x = \{x_1, x_2, ..., x_n\}$ и возможно размерность n, причем размерность и состав могут быть определены только динамически в ходе функционирования.

Таким образом, гибридная система является «склейкой» чисто непрерывных систем $\{C_1, C_2, ..., C_m\}$ причем решение $x_j(t_j^* - 0)$ системы C_j с учетом последовательности мгновенных действий A^* является начальными условиями x_{j+1}^0 для системы C_{j+1}. В частном случае $C_j = C_{j+1}$. Область существования системы C_j в расширенном фазовом пространстве (x,t) размерности $n+1$ задается некоторым логическим предикатом $P_j(x,t)$, который принимает значение false внутри области существования C_j и true вне нее. Таким образом, время t_j^* очередного дискретного события определяется моментом изменения значения предиката $P_j(x,t)$ с false на true.

Может случиться так, что вычисленное начальное значение x_{j+1}^0 для непрерывной системы C_{j+1} таково, что предикат $P_{j+1}(x_{j+1}^0, t^*) = true$ и следовательно в тот же момент непрерывного времени t^* произойдет следующее дискретное событие. В общем случае в момент t^* может произойти цепочка дискретных событий $E_1^* \to E_2^* \to ... \to E_m^*$. Несмотря на то, что в непрерывном времени все эти события одновременны, эта цепочка имеет очевидную причинно-следственную упорядоченность и можно говорить о локальном для момента t^* дискретном времени $k = 1,2,...,m$. Такие моменты t^* принято называть «временной щелью» (time gap) [110,111]. Внутри такой «временной щели» можно говорить о гибридном времени (t,k).

Гибридные модели в инструментальных средствах для «больших» ЭВМ.

В данном разделе рассматриваются несколько подходов к моделированию гибридных систем, типичных для «ранних» инструментов компьютерного моделирования (до начала 1990-х гг.). Характерными чертами этих инструментов является их «невизуальность», то есть ориентация на текстовое

представление описания модели и пакетный режим работы компьютера типа IBM-370 или БЭСМ-6, а также построение входного языка как проблемно-ориентированного расширения известного языка программирования. Однако, несмотря на кажущуюся архаичность, в этих инструментах уже присутствуют легко узнаваемые черты современных подходов к компьютерному моделированию гибридных систем.

Язык SLAM II.

Этот язык имитационного моделирования [50] является одним из последних и самых мощных языков моделирования, построенных как расширение популярного языка программирования вычислений Фортран. Язык SLAM II вобрал в себя лучшие решения более ранних разработок, таких как GPSS, SIMSCRIPT, GASP IV и включает в себя средства для моделирования дискретных, непрерывных и непрерывно-дискретных систем.

Средства дискретного моделирования включают в себя средства так называемого «сетевого моделирования», ориентированные на системы массового обслуживания и использующие традиционные для этого направления понятия транзакции, очереди, ресурса и т.п., а также «дискретно-событийные» средства моделирования, позволяющие задавать дискретные последовательные процессы, развивающиеся в непрерывном времени и взаимодействующие между собой через дискретные события, В языке различаются «временные события» и «события-состояния», которые могут планироваться пользователем. «Временное событие» происходит по истечении указанного интервала модельного времени, «Событие-состояние» происходит при пересечении значением указанной переменной определенного порогового значения, причем можно указать направление пересечения (положительное или отрицательное) и точность по значению переменной. Пользователь может «подвесить» на дискретное событие «обработчик», то есть некоторую последовательность действий, которая выполняется в модельном вре-

мени мгновенно. Эта последовательность может изменять значения переменных модели, а также планировать новые события.

Средства непрерывного моделирования позволяют задавать системы обыкновенных дифференциальных уравнений первого порядка в форме Коши, а также наборы формул (транслятор не выполняет автоматической сортировки формул и правильность порядка их записи возлагается на пользователя). Численное интегрирование осуществляется методом Рунге-Кутта-Фелберга с переменным шагом и контролем точности. Непрерывные переменные могут определять «события-состояния» и таким образом непрерывная составляющая модели может генерировать дискретные события. В «подвешенной» на эти события последовательности действий значения непрерывных и дискретных переменных могут меняться скачками. Это не приводит к неправильным результатам численного интегрирования, так как исполняющая система SLAM II начинает процесс интегрирования заново после каждого дискретного события. В обработчике события могут также изменяться значения специальных переменных-переключателей, в зависимости от которых значения правых частей дифференциальных уравнений и формул могут вычисляться по различным алгоритмам (то есть задаваться т.н. условные или гибридные уравнения в современной терминологии). Непрерывная и дискретная составляющая модели могут влиять друг на друга через дискретные события, изменяющие значения общих переменных.

Таким образом, если отвлечься от архаического способа задания описания модели на языке Фортран, язык SLAM II предоставляет достаточно мощные средства для моделирования непрерывно-дискретных систем. Язык позволяет задавать качественные изменения типа 1 (скачкообразные изменения значений непрерывных переменных), а также типа 2 (изменение отображения C при неизменной структуре вектора x) и это делается корректно с точки зрения численного решения. Изменения типа 3 и 4 (изменение структуры вектора x) в языке SLAM II задать невозможно. Вид непрерывного отображения C весьма ограничен. К недостаткам языка также следует отне-

сти низкую наглядность использования переменных – переключателей для изменения алгоритмов вычисления правых частей уравнений, а также взаимосвязей обработчиков дискретных событий. Опыт показывает, что при разработке программ в аналогичном стиле (например, при программировании оконных сообщений MS Windows) разработчики делают очень много ошибок.

Язык НЕДИС.

Язык моделирования непрерывно-дискретных систем НЕДИС [16] является расширением объектно-ориентированного языка Simula-67 [19]. Этот язык является примером подхода, в котором входной язык пользователя и язык исполняющей системы пакета моделирования совпадают. Так, в исполняющей системе НЕДИС переопределяется заданный в Simula-67 стандартный класс SIMULATION для того, чтобы иметь возможность моделировать непрерывную составляющую поведения, а пользователь может переопределить класс INTGRL для того, чтобы использовать свой собственный численный метод интегрирования.

Дискретное поведение задается с помощью последовательного процесса как цепочки дискретных событий. Под дискретным событием понимается мгновенно исполняемая последовательность операторов. Таким образом, текстуальное описание последовательного процесса представляет собой последовательности операторов, соответствующих дискретным событиям, разделенные специальными операторами синхронизации. Очередное дискретное событие может быть привязано к определенному моменту модельного времени или к выполнению некоторого условия. Кроме того, одни процессы могут активизировать и пассивизировать другие. Так как язык является объектно-ориентированым, то описание процесса задается в виде описания класса, а конкретный процесс появляется как экземпляр этого класса после обращения к конструктору класса с набором действительных параметров.

Неявно предполагается, что существует некоторая глобальная для модели в целом система обыкновенных дифференциальных уравнений первого порядка Новое уравнение автоматически добавляется в глобальную систему уравнений при вызове конструктора предопределенного класса INTGRL и убирается из глобальной системы уравнений при вызове деструктора. В действительных параметрах конструктора класса INTGRL указывается начальное значение интегрируемой переменной, а также ссылка на функцию, определяющую правую часть уравнения.

Таким образом, непрерывный объект появляется в модели когда какой-нибудь последовательный процесс в точке очередного дискретного события создаст экземпляры дифференциальных уравнений, соответствующих поведению этого объекта. Полученное совокупное непрерывное поведение модели будет определять значения непрерывных переменных до следующего дискретного события. Изменение непрерывных переменных может вызвать очередное дискретное событие, заданное по условию. Используя возможности синхронизации последовательных процессов и операторы ветвления алгоритмов, можно задать практически любую логику изменения непрерывного поведения.

Интересной особенностью языка является то, что алгоритм главного процесса модели по существу является планом вычислительного эксперимента, позволяющим не только получать отдельные фазовые траектории модели, но и строить параметрические зависимости и т.п.

Таким образом, теоретически на языке НЕДИС можно создать гибридную модель любого типа, в том числе и с переменной размерностью фазового вектора. Однако, это достигается во-первых, за счет серьезных ограничений на непрерывное отображение C, а во-вторых, за счет очень низкого уровня конструкций языка. Пользователю практически предлагается вручную обращаться к входам исполняющей системы пакета. Даже несложный пример с моделью чисто непрерывной системы автоматического регулиро-

вания, приведенный в [16], понять практически невозможно без соответствующей структурной схемы.

Гибридные модели в современных инструментах моделирования.

Современные инструменты моделирования непрерывно-дискретных систем можно разделить на три группы:

- инструменты «блочного моделирования»;
- инструменты «физического моделирования».
- инструменты, использующие формализм гибридного автомата.

. В Приложении 1 рассматривается реализация нескольких типовых примеров гибридных моделей, созданных с использованием различных подходов.

Гибридное моделирование в пакете SIMULINK.

Для задания непрерывной части модели предназначены стандартные блоки, содержащихся в библиотеке «Continuous» (интегратор, дифференциатор, передаточная функция и др.). С их помощью, а также используя библиотеку «Math», содержащую блоки, соответствующие стандартным математическим операциям и элементарным функциям, можно собрать блок-схему, соответствующую любой системе обыкновенных дифференциальных уравнений в форме Коши и формул.

Для задания дискретной части модели предназначены стандартные блоки, содержащиеся в библиотеке «Discrete». В библиотеке содержатся блоки, которые позволяют решать разностные уравнения, описывающие системы, с помощью дискретного преобразования Лапласа или так называемого z-преобразования. Сигналы на выходе дискретных блоков являются кусочно-постоянными функциями времени (Рис. 6), что позволяет соединять их с непрерывными блоками. В библиотеках «Nonlinear» и «Sources» содержится достаточно богатый набор стандартных гибридных блоков, таких как звено с насыщением, зона нечувствительности, люфт, петля гистерезиса, генератор

прямоугольных импульсов, генератор пилообразного сигнала и др. При использовании этих блоков следует надеяться, что исполняющая система SIMULINK начинает численное интегрирование заново на каждом переключении в этих стандартных блоках. Для моделирования нестандартных случаев следует использовать ряд специальных блоков: блок «Switch» из библиотеки «Nonlinear», блоки «Enable», «Trigger» и «HitCrossing» из библиотеки «Signals&Systems». Кроме того, для этих целей предусмотрены дополнительные входы и режимы работы блока «Integrator».

Для моделирования дискретных скачков значений переменных можно использовать специальные входы сброса интегратора и повторной инициализации значения интегрируемой переменной (см. пример 1 в Приложении 1 – модель мячика, отскакивающего от плоскости). Для этой же цели можно использовать блок «Switch», который по изменению знака управляющего входа подает на выход значения первого или второго входов.

Для моделирования изменения непрерывного отображения C необходимо создать блок-схемы, соответствующие всем возможным системам уравнений. Кроме того, необходимо также вырабатывать специальный управляющий сигнал, знак которого изменяется в момент появления дискретного события, и подавать его на управляющий вход блока «Switch», который переключает соответствующие ветви блок-схемы (см. пример 2 в Приложении 1 – модель мячика, падающего на вертикальную пружину). Для большей понятности схемы участки блок-схемы, соответствующие различным вариантам системы уравнений целесообразно оформлять как подсистемы, то есть создавать иерархическую блок-схему (см. пример 3 в Приложении 1 – модель отрывающегося маятника).

В изложенном выше способе скрыта определенная опасность: может оказаться, что все отображения C просто не определены на всем фазовом пространстве (например, блок $y = \sqrt{x}$ выдаст ошибку при отрицательном значении входа). Для того, чтобы этого не происходило, необходимо воспользоваться блоком «Enable». Этот блок, помещенный в подсистему, включает ее

при положительном значении управляющего входа и отключает при отрицательном или нулевом значении управляющего входа. В зависимости от установок блока выходы подсистемы после отключения могут сохранять последнее значение, вычисленное во включенном состоянии, или же инициализироваться начальными значениями. Использование блока «Enable» показано в примере 3 Приложения 1.

Таким образом, стандартными средствами SIMULINK можно создавать довольно сложные гибридные модели. Возникающие неудобства пользователя связаны скорее не с гибридной моделью, а с неудобством вообще набора сложных систем уравнений из примитивных блоков. Общим принципом гибридного моделирования в SIMULINK является использование готовых гибридных блоков или переключение заранее заготовленных альтернативных участков блок-схем. Ясно, что методом переключения ветвей блок-схемы принципиально невозможно моделировать системы с динамически изменяемой структурой. Возникают значительные трудности, связанные с описанием мгновенных действий при обработке дискретного события и тем более при описании цепочки дискретных событий во временной щели. (решения типа сброса интегратора следует признать все же искусственными). Кроме того, запутанные переключательные схемы чрезвычайно сложны для понимания.

Для того, чтобы помочь пользователю преодолеть эти трудности, в последние версии SIMULINK введена специальная подсистема STATEFLOW, позволяющая создавать специальные дискретные блоки, функционирование которых задается картой состояний Харела. Блок «STATEFLOW,» оформляется как подсистема и через свои входы и выходы может взаимодействовать с обычной блок-схемой. Безусловно, подсистема STATEFLOW существенно облегчает разработку нестандартных дискретных подсистем. Карта состояний позволяет легко задавать любую сложную логику возникновения дискретных событий и их обработки, а также имеет очень наглядное визуальное представление. Однако, опыт показывает (подобное решение было использовано автором в пакете Model Vision 2.1 [34]), что независимое параллельное

функционирование дискретной карты состояний и непрерывного поведения, взаимодействующих только через общие переменные, все равно чрезвычайно сложно для восприятия рядовым пользователем и часто приводит к трудно выявляемым ошибкам.

Таким образом, подход «блочного моделирования» не отвечает требованиям, предъявляемым к средствам автоматизации моделирования сложных СУ:

- принципиально невозможно моделирование систем с динамической структурой;
- описание сложного непрерывного поведения приводит к запутанным и неестественным схемам;
- невозможно полноценное объектно-ориентированное моделирование;
- поддержка компонентного моделирования с ненаправленными связями возможна только для специальных наборов блоков.

Гибридное моделирование на языке Modelica.

Общим принципом гибридного моделирования в языке Modelica является т.н. «принцип синхронного потока данных» («synchronous data flow principle») [113]. Согласно этому принципу непрерывная часть модели представляется совокупной системой алгебро-дифференциальных уравнений, а дискретная часть модели трактуется как набор дополнительных уравнений, которые присоединяются к совокупной системе уравнений непрерывной части в момент возникновения дискретного события (принцип соответствует схеме на Рис. 7).

Обработчик дискретного события (блок `when`) представляет собой набор уравнений (их тип ограничен – это могут быть только формулы), добавляемые к текущей совокупной системе уравнений только в момент возникновения ассоциированного с этим блоком дискретного события. В общем случае обработчик ассоциируется с дискретным событием через определенный

логический предикат – событие происходит в момент переключения значения этого предиката с false на true. Предикат для периодических событий, соответствующих разностному уравнению, выделяется как особая предопределенная функция. Декларируется, что исполняющая система должна прекращать численное интегрирование при возникновении дискретного события и затем продолжить его с новыми начальными значениями.

Все переменные делятся на непрерывные и дискретные. Непрерывные переменные изменяются между дискретными событиями, а дискретные только в обработчике дискретного события. Значения дискретных переменных сохраняются неизменными до следующего события (Рис. 6). Изменение значения непрерывной переменной в обработчике дискретного события возможно только через специальную предопределенную процедуру (вспомним специальный вход интегратора в SIMULINK!).

В результате изменений дискретных переменных в одном обработчике дискретного события может стать истинным предикат другого обработчика и таким образом может возникнуть цепочка дискретных событий. Это очень похоже на задание дискретных процессов в языке SLAM II и столь же неналядно. Очевидное неудобство задания сложной дискретной логики заставляют делать попытки частичной реализации карт состояний в виде макронадстройки над языком Modelica [109,89] (напрашивается очевидная аналогия с подсистемой STATEFLOW, добавленной в SIMULINK). С помощью обработчиков дискретных событий достаточно просто описывается гибридное поведение первого типа (например, отскок мячика от плоскости в примере 1 Приложения 1):

Modelica позволяет использовать условные выражения в правых частях уравнений. Такие уравнения называют условными или гибридными. Таким образом, если размерность фазового вектора и соответственно число уравнений не меняются (гибридное поведение второго типа), то изменение отображения C задается просто и изящно. Например, запись примера 2 из Приложения 1 (мячик, падающий на вертикальную пружину) выглядит крат-

кой и понятной по сравнению с громоздкой блок-схемой, реализующей эту же модель в SIMULINK.

Для случая, когда размерность фазового вектора меняется, дело обстоит хуже. Язык запрещает делать это во время выполнения модели. Руководство [108] рекомендует для тех случаев, когда структура фазового вектора изменяется, но размерность сохраняется, просто использовать старые переменные, нагружая их другим смыслом. В случае, когда размерность меняется (пример 3 из Приложения 1 – обрывающийся маятник), рекомендуется создавать для каждого варианта непрерывного поведения свой локальный объект (в данном примере это колеблющийся маятник и оторвавшийся маятник, и по дискретному событию включать следующий объект (у всех объектов языка Modelica имеется предопределенный параметр `enable`), инициализировать его переменные текущими значениями предыдущего объекта и выключать предыдущий объект. Эта методика почти дословно совпадает с методикой SIMULINK. Естественно, при таких ограничениях моделирование систем типа 4 с динамической структурой на языке Modelica принципиально невозможно.

Эти ограничения связаны с тем, что поведения всех экземпляров всех классов, входящих в модель, на этапе компиляции превращаются (это декларировано на уровне спецификации языка [107]) в эквивалентную одноуровневую систему уравнений следующего вида:

$$v := [\dot{x}'; x; y; t; m; pre(m); p]$$
$$c := f_c(relation(v))$$
$$m := f_m(v, c)$$
$$0 = f_x(v, c)$$

где

p - параметры и константы;

t - независимое непрерывное время;

$x(t)$ - дифференциальные переменные;

$m(t_e)$ - дискретные переменные, которые изменяются только в моменты t_e;

$y(t)$ - алгебраические переменные;

$c(t_e)$ - условия всех операторов `if` и `when`;

$relation(v)$ - отношения между компонентами v_i вида $v_1 > v_2, v_3 \geq 0$ и т.п.

Эта система содержит алгебраические уравнения булевского типа и ее решение является весьма непростой задачей, требующей разработки специальных численных методов типа LSODAR [59,72]. Возникает естественный вопрос: зачем авторам языка Modelica обязательно нужно получать эту систему уравнений и затем преодолевать трудности ее численного решения?

Скорее всего, имеются две причины. Во-первых, это традиционный подход для моделирования в тех прикладных областях, из которых «выросла» Modelica. Ясно, что разработчики пакета Dymola, поддерживающего язык Modelica, опирались на ранее созданный задел в части компилятора и исполняющей системы. Во-вторых, принятая в языке свобода записи уравнений плюс ненаправленные связи между компонентами приводят к возникновению непростых проблем при объединении описаний отдельных компонент в одну эквивалентную систему уравнений. Эти проблемы связаны с анализом корректности системы, поиском набора искомых переменных, поиском согласованных начальных значений и др. Этих проблем нет в языках SLAM II, НЕДИС и SIMULINK, так как там введены жесткие ограничения на форму задания непрерывного поведения и связей между компонентами. Такие ограничения принципиально неприемлемы для декларированной области применения «моделирование физических систем». Традиционно принято переносить выполнение такого сложного анализа на стадию компиляции и избегать его на стадии выполнения модели. Проблема усугубляется еще и тем, что иногда необходимо выполнять дифференцирование отдельных уравнений [105]. Поскольку выполнять численное дифференцирование крайне нежелательно, необходимо выполнять аналитическое дифференцирование исходных

уравнений. Кроме того, для ряда случаев чрезвычайно полезно аналитическое вычисление матрицы Якоби. Эти вычисления также обычно выполняются на стадии компиляции.

Таким образом, подход «физического моделирования», основанный на языке Modelica, также не соответствует требованием, предъявляемым к средствам автоматизации моделирования сложных СУ:

- принципиально невозможно моделировать системы с динамической структурой;
- описание дискретных действий чрезвычайно ненаглядно и не соответствует языку UML..

Гибридные модели на базе формализма «гибридный автомат».

Формальная схема так называемого гибридного автомата активно используется в работах по качественному анализу гибридных систем [98,101,103].

Гибридным автоматом называется совокупность

$H = \{t, G, V, C, P, A, F\}$, где

- $G = \{S, s_0, E\}$ - ориентированный граф, вершины которого сопоставлены элементам множества дискретных состояний автомата $S = \{s_i \mid i = 1..m\}$, а дуги – возможным переходам автомата из одного состояния в другое $E : S \to S$. Одно из состояний s_0 является начальным.

- $t \in T \subset \Re$ - независимая переменная, определяющее значение непрерывного времени;

- $V = \{V_C, V_D\}$ - множество переменных, в том числе $V_C = \{v_i \in \Re \mid i = 1..n_C\}$ - множество непрерывных переменных и $V_D = \{v_i \in \Re \cup Z \cup B \mid i = 1..n_D\}$ - множество дискретных переменных ($B = \{false, true\}$ - множество логических значений).

- $C = \{c_i : (t, V) \to V_C \mid i = 1..k_C\}$ - множество непрерывных отображений;

- $P = \{p_i(t, V) \in B \mid i = 1..k_P\}$ - множество логических предикатов.

- $A = \{a_i : V \to V \mid i = 1..k_A\}$ - множество мгновенных действий.

- $F = \{F_C, F_P, F_A\}$, где $F_C : C \to S$ - отображение, сопоставляющее множество непрерывных отображений множеству состояний (вершин графа), $F_P : P \to E$ - отображение, сопоставляющее множество предикатов множеству переходов (дуг графа), $F_A : A \to E$ - отображение, сопоставляющее множество мгновенных действий множеству переходов (дуг графа).

Таким образом, гибридный автомат представляется ориентированным графом, вершинам которого сопоставляются некие качественные состояния системы и приписываются непрерывные действия, которые выполняются пока данное состояние является текущим (конкретные формы непрерывных действий в работах по анализу гибридных систем либо не уточняются, либо под ними понимаются формулы или линейные дифференциальные уравнения). В момент $t = 0$ текущим является состояние s_0 (соответствующая вершина графа помечается специальным маркером). Далее гибридный автомат функционирует следующим образом. Переменная t изменяется независимо (это принципиальное отличие от обычного конечного автомата) и выполняются непрерывные действия $c_i = F_C(s_i)$, приписанные текущему состоянию s_i. Если состоянию не приписано никакого непрерывного отображения (состояние пустое), то не выполняется никаких действий и все переменные сохраняют свои значения. Это происходит до тех пор, пока для какого либо перехода e_{ij} из состояния s_i в состояние s_j приписанный ему логический предикат $p_{ij} = F_P(e_{ij})$ не примет значение *true* (если переходу не приписано никакого предиката, то предполагается, что ему приписан предикат со значением *true*). В этот момент t^* непрерывные действия c_i в состоянии s_i перестанут выполняться, непрерывные переменные сохранят свои последние вычисленные значения, сработает переход e_{ij}, выполнятся приписанные ему мгновенные действия $a_{ij} = F_A(e_{ij})$, изменяющие значения некоторых непрерывных и дискретных переменных и текущим станет состояние s_j. Далее начнет снова

продвигаться непрерывное время t и выполняться приписанное ему непрерывное отображение $c_j = F_C(s_j)$ с начальными значениями $V(t^*)$.

Иногда в определение гибридного автомата добавляют специальный инвариант $Inv(V,t)$, определенный для каждого состояния автомата [34]. Значение инварианта всегда должно быть *true*, в противном случае возникает исключительная ситуация и автомат прекращает функционирование.

На Рис. 8 показан гибридный автомат, моделирующий процесс катапультирования пилота из самолета, движущегося горизонтально со скоростью V_p (модель заимствована из [50]).

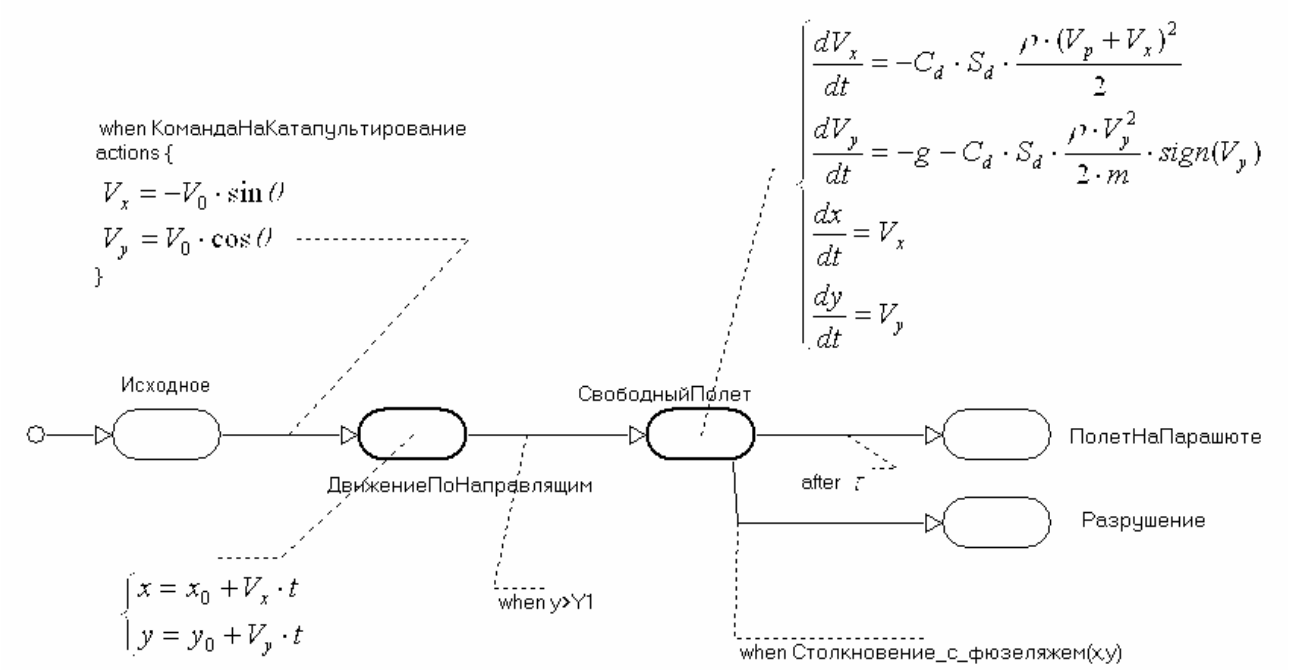


Рис. 8

Граф переходов автомата включает в себя 5 состояний и 4 перехода. Начальным состоянием является `Исходное`, которому не приписано никакого непрерывного отображения. Автомат выходит из этого состояния, когда булевская переменная `КомандаНаКатапультирование` принимает значение *true*. После начала катапультирования кресло с пилотом начинает двигаться по направляющим с постоянной скоростью V_0 под определенным углом θ к вертикали. Этому процессу соответствует состояние `ДвижениеПоНаправляющим`, которому приписано непрерывное отображение, состоящее из двух формул . Постоянные значения вертикальной и горизонтальной составляющих скорости рассчитываются в мгновенных действиях перехода (для данного состояния переменные

V_x и V_y являются дискретными). После подъема на некоторую высоту *Y1* кресло сходит с направляющих и переходит в фазу свободного полета, в которой траектория движения определяется действием силы тяжести и сопротивления воздуха. Этой фазе соответствует состояние `СвбодныйПолет`, которому приписана система из четырех дифференциальных уравнений первого порядка. Свободный полет завершается либо раскрытием парашюта через время τ , либо аварией (столкновением с корпусом самолета

Из приведенного выше примера (а также других, приведенных в Приложении 1) видно, что формальная схема гибридного автомата:

- позволяет в очень компактной и наглядной форме описывать как мгновенные изменения значений переменных, так и любые изменения непрерывного поведения и размерности фазового вектора;
- позволяет задавать цепочки чисто дискретных действий с помощью состояний без непрерывных отображений, что соответствует машине состояний UML;
- позволяет задавать компонентную структуру модели (как статическую, так и динамическую) в виде параллельно функционирующих гибридных автоматов;
- явно отделяет непрерывные аспекты поведения от дискретных, что облегчает работу численных методов.

Все это говорит о том, что гибридный автомат является чрезвычайно привлекательной математической моделью гибридной системы для построения на ее основе инструмента компьютерного моделирования. Как ни странно, на основе этой модели построен только язык SHIFT [85]. Этот язык не имеет визуального представления, поддерживает только простейшие виды непрерывного поведения (дифференциальные уравнения в форме Коши и правильная последовательность формул) и ориентирован использование в рамках среды SmartAHS, предназначенной для анализа дорожного движения [92]. Любопытно, что разработчики этого проекта осознано отказались от использования SIMULINK и Modelica именно из-за невозможности моделиро-

вать динамические структуры (что очевидно необходимо для модели системы массового обслуживания) и были вынуждены создать свой язык, выбрав в качестве базовой формальную схему гибридного автомата.

Таким образом, представляется, что подход, использующий формализм гибридного автомата, в принципе соответствует всем предъявляемым требованиям и является наиболее перспективным для средств автоматизации моделирования сложных динамических систем.

Существующие языки объектно-ориентированного моделирования.

Объектно-ориентированный подход позволяет задавать моделируемую систему в виде совокупности объектов. Каждый объект всегда является экземпляром какого-то класса со своими собственными значениями параметров. Определение класса задает прототип объекта. Иногда предусматривается параметризация самого определения класса. Другими составляющими объектно-ориентированного подхода являются наследование классов и полиморфизм экземпляров объектов. Для объектно-ориентированного моделирования какого-то конкретного вида систем, например, гибридных, необходимо определить, что считается объектом, определить способы параметризации объектов, а также определить, что конкретно означают наследование и полиморфизм для объектов данного вида. Ниже показано, как это делается в ряде существующих языков объектно-ориентированного моделирования.

Simula-67 и НЕДИС.

В языке Simula-67 [19] под объектом понимается совокупность переменных и методов (функций и процедур, то есть традиционный объект в языках программирования.. Кроме того, с объектом может быть связан процесс, выполняемый параллельно с процессами в других объектах. Под процессом понимается просто последовательность операторов, включающая специальные операторы синхронизации. Таким образом, уже в этом языке появляется деление объектов на «активные» (объект, имеющий свою собственную «нит-

ку управления) и ««пассивные» (методы «пассивного» объекта могут быть вызваны из «активных» объектов или из главной программы). Предполагалось, что построив подходящие наборы стандартных классов, можно будет на базе Simula-67 создавать специализированные языки моделирования. Подкласс наследует все атрибуты и методы суперкласса, виртуальные методы могут быть переопределены. Унаследованное дискретное поведение (процесс), таким образом, может быть переопределено с помощью переопределения используемых в теле процесса процедур и функций.

Язык НЕДИС [16] является расширением Simula-67 и позволяет наряду с дискретным поведением моделировать также и непрерывное поведение. Дифференциальные уравнения задаются в форме явного вызова функции интегрирования с указанием правой части уравнения, заданной в виде функции. Таким образом, подкласс наследует непрерывное поведение суперкласса. С помощью переопределения соответствующих функций можно в подклассе переопределять уравнения, заданные в суперклассе. Возможны как статические, так и динамические экземпляры объектов.

ObjectMath.

Язык ObjectMath [91,117] является расширением входного языка пакета Mathematica [21]. Основной концепцией этого языка является поддержка «объектно-ориентированного математического моделирования», противопоставляемого объектно-ориентированному программированию. Под объектом в ObjectMath понимается совокупность переменных, процедур и функций, а также уравнений (в синтаксисе Mathematica). Некоторые переменные, объявленные как внешние, доступны извне объекта. Взаимодействие объектов осуществляется через уравнения объекта-контейнера, которые играют роль уравнений связей между внешними переменными локальных объектов. Подкласс наследует все элементы суперкласса (включая уравнения). В подклассе могут быть добавлены новые переменные, функции и уравнения, а также пе-

реопределены унаследованные функции и уравнения. Все экземпляры объектов статические.

Omola.

Несмотря на то, что язык Omola [69,68] как бы «влился» в язык Modelica, мы его рассмотрим отдельно, поскольку он имеет ряд интересных особенностей, которых нет в Modelica. Прежде всего, в Omola различают два вида объектов: «полноценные» объекты и «объекты с семантикой значения», подобные структурам в языке C# [43]. Основное различие между этими видами объектов состоит в том, что для объектов первого вида имя объекта понимается как указатель на данный экземпляр, а для объектов второго вида оно понимается как значение. Определения классов можно группировать в библиотеки классов. Имеется ряд «базовых» классов, определенных в библиотеке `Base`. «Полноценный» объект в Omola ассоциируется с параллельно функционирующим компонентом модели. Компонент представляет собой совокупность атрибутов. Атрибутами могут быть:

- локальные объекты;
- определения локальных классов;
- простые переменные вещественного, целого, булевского и строкового типа, а также матрицы;
- уравнения;
- формулы;
- связи.

К локальным объектам относятся:

- локальные компоненты (элементы локальной блок-схемы, которые, возможно, связаны связями);
- параметры;
- переменные-объекты;

- «терминалы», то есть внешние переменные, через которые осуществляются связи между компонентами;
- дискретные события.

Определение локального класса находятся внутри определений другого класса и в нем видимы все атрибуты этого класса.

Подкласс наследует все атрибуты суперкласса. В подклассе могут быть добавлены любые новые атрибуты. Переопределение атрибута суперкласса осуществляется простым и изящным способом: любой атрибут подкласса с именем, совпадающим с именем атрибута суперкласса, переопределяет последний. Таким способом можно, например, изменить тип значения унаследованной переменной. Например, атрибут `value` в базовом классе `Variable` имеет вещественный тип, но в любом потомке класса `Variable` можно объявить его с другим типом значения. Таким образом, нет необходимости в использовании параметризованных классов. Поскольку для переопределения необходимо имя атрибута, не могут быть переопределены уравнения и связи. Формулы могут быть переопределены, поскольку они рассматриваются как дополнение («связывание») к определению переменной, стоящей в левой части формулы. Такой подход к переопределению атрибутов противоречит, правда, общепринятому подходу ООП, в котором атрибут с совпадающим именем лишь замещает атрибут суперкласса в теле подкласса (но не в теле суперкласса!).

Еще одной интересной особенностью Omola является возможность указания в атрибуте `quantity` переменной-объекта физической сущности переменной из некоторого списка (угол, расстояние, масса, температура и т.п.), а в атрибуте `unit` единиц измерения.

Modelica.

Язык Modelica [106,107,108] в настоящее время является самым «продвинутым» языком ООМ. В этом языке декларируется, что классом является практически любое определение (даже алгоритмическая функция!), вследст-

вие чего структура классов получается довольно запутанной. Выделяются важные семантические разновидности классов («ограниченные классы» в терминологии Modelica или стереотипы классов в терминологии UML):

- `model` – неориентированный компонент, который может содержать параметры, переменные, коннекторы, локальные компоненты, уравнения (в том числе уравнения связей), алгоритмы;
- `block` - ориентированный компонент, все внешние переменные которого должны быть либо входами, либо выходами;
- `connector` - внешняя переменная, которая может участвовать в связях, не может содержать уравнений;
- `record` – определение записи;
- `type` – определение типа пользователя, может являться расширением предопределенных типов, записи, перечислимого типа и массива;
- `package` – пакет (библиотека классов), может содержать только определения классов и констант;
- `function` – алгоритмическая функция.

Неявно предполагается, что все классы стереотипа «type» являются классами «объектов с семантикой значения».

Атрибуты компонента могут быть параметрами, переменными, коннекторами, компонентами, функциями или константами. Атрибуты могут быть видимыми извне (по умолчанию) или инкапсулированными в описании компонента. В связях могут участвовать только коннекторы. Таким образом, компоненты могут взаимодействовать между собой по связям через коннекторы, а также через уравнений охватывающего компонента, в которые входят видимые извне атрибуты. Для взаимодействий вида «один со всеми» или «все со всеми», характерных для задач, в которых фигурируют физические поля, предусмотрен специальный механизм inner/outer-атрибутов. Этот механизм заключается в следующем: если в каком-либо компоненте модели объявлен атрибут со статусом `outer`, то он рассматривается как ссылка на неко-

торый атрибут, имеющий то же самое имя и совместимый тип, но объявленный в каком-либо из охватывающих компонентов и имеющий статус `inner`. Например, в определении класса, задающего свободное движение материальной точки, можно не детализировать зависимость ускорения силы тяжести от высоты, предполагая, что эта функция будет определена в охватывающем компоненте, задающем правила вычислительного эксперимента.

Подкласс наследует все элементы описания суперкласса. Ни удалить, ни переопределить унаследованные элементы описания нельзя. Например, если вы хотите использовать модель сопротивления, в которой учитывается влияние температуры, вы не сможете ввести класс «термосопротивление» как потомок класса «сопротивление», поскольку для этого вам пришлось бы заменить уравнение $R \cdot I = V$ другим уравнением $(R_0 + R_T \cdot (T - T_0)) \cdot I = V$. В подклассе можно модифицировать лишь начальные значения переменных, значения параметров по умолчанию и значения констант. Помимо понятий «подкласс» и «суперкласс» вводятся также понятия «подтип» и «супертип». Класс A является подтипом класса B, если содержит все видимые извне атрибуты класса B (совпадение по именам) и типы этих атрибутов являются подтипами соответствующих типов атрибутов класса B. Это очень похоже на отношение `implements` между классом и интерфейсом в языке Java, если под интерфейсом понимать совокупность методов и переменных. Для функций совместимость по интерфейсу означает наличие одноименных параметров с совместимыми типами и совместимость типов возвращаемого значения. Объект класса B может быть заменен на объект класса A даже если класс A не является производным от B. Это дает возможность параметризации класса, которая позволяет переопределять переменные при наследовании этого класса. Например, вы можете определить класс C1, задающий некоторую схему, в которой используется обычное сопротивление `R1`. Затем вы можете создать производный от него класс C2, в котором компонент `R1` будет заменен на термосопротивление, или на конденсатор, или на индуктивность. Эти замещения будут корректными, поскольку все замещающие и замещаемый

классы по внешнему интерфейсу являются подтипами типа «двухполюсник». Механизм замещения может применяться не только к атрибутам-экземплярам, но и к атрибутам-классам. Таким образом, с помощью замещения создается возможность параметризации классов. Правда, атрибуты, которые могут быть замещены, должны быть специально помечены в определении класса.

Для объединения элементов описания в группы (библиотеки классов) используются пакеты. Пакет — это контейнер для группы компонентов, ограничивающий область их видимости. Компоненты, объявленные как `public`, видимы извне под составным именем, включающим в качестве префикса имя пакета. Остальные компоненты видимы только внутри данного пакета. В отличие от языков программирования, где компонентами пакета являются только классы, естественными компонентами пакета в ООМ являются также константы и алгоритмические функции. Для того, чтобы пакет был виден в другом пакете или модели, его нужно импортировать, то есть указать явным образом на его использование. Импортирование пакета означает, что его имя становится видимым в импортирующем пакете или модели. Можно импортировать не весь пакет, а только конкретные компоненты пакета. В качестве компонента пакета может выступать другой пакет. Относительно видимости компонентов пакета во вложенных пакетах имеется два решения: по умолчанию все компоненты охватывающего пакета видны во вложенном пакете, в пакетах со статусом `encapsulated` в пакете видимы только собственные и явно импортируемые компоненты.

Объектно-ориентированное моделирование карт состояний.

Объекты, поведение которых задается картами состояний, называются в UML [12] активными. Состояниям активных объектов могут приписываться некоторые деятельности, имеющие ненулевую длительность. Хотя под деятельностью понимается лишь последовательность (возможно циклическая) чисто дискретных операций (т.н. «нитка управления»), ясно, что эти

объекты представляют интерес для данного исследования, поскольку гибридный автомат является естественным обобщением активного объекта UML. Вопросы наследования и полиморфизма для активных объектов рассматриваются в работе [94] применительно к инструментальной системе Rhapsody. Производный класс наследует все переменные и карты состояний базового класса. В производном классе могут быть добавлены новые переменные, новые карты состояния, а также новые состояния и переходы в унаследованных картах состояний. Новая переменная с тем же именем, что и определенная в базовом классе, замещает ее только в описании производного класса. Кроме того, в унаследованных переходах можно переопределить условия срабатывания и действия, а в унаследованных состояниях можно переопределить входные и выходные действия. В работе [94] рекомендуется проводить специализацию поведения производного класса преимущественно путем добавления параллельных карт состояний.

Инструменты «блочного моделирования».

Инструменты этого направления предоставляют очень ограниченные возможности ООМ. Библиотеки стандартных блоков являются библиотеками классов. Пользователь может поместить экземпляр стандартного класса в модель и изменить значения его параметров. Однако, определить новый класс в рамках входного языка можно только как подсистему, поведение которой задается структурной схемой.

Анализ существующих языков ООМ применительно к моделированию сложных динамических систем.

В рассмотренных языках ООМ можно выделить несколько характерных особенностей, существенных для моделирования сложных динамических систем.

При моделировании непрерывных систем в качестве интерфейса объекта рассматривается совокупность внешних переменных, а не совокупность методов и сообщений, как в языках объектно-ориентированного программи-

рования и UML. Это отражает объективные особенности функционирования непрерывных систем. Следовательно, в языке моделирования сложных динамических систем, ориентирующемся на UML, понятие активного объекта должно быть расширено.

Под объектом понимается структурный компонент моделируемой системы, функционирующий параллельно в модельном времени с другими структурными компонентами. Существующие языки ООМ ориентированы в значительной степени на технологию «промышленного» моделирования, предполагающую преимущественное использование в конкретных проектах заранее разработанных прикладных библиотек классов. Предполагается, что определения классов в основном будут писаться высококвалифицированными разработчиками библиотек, а обычные пользователи будут строить модель в виде совокупности стандартных объектов, взаимодействующих явно через связи или неявно через уравнения или дискретные действия объекта-контейнера В этом смысле подходы SIMULINK и Dymola очень близки, только Dymola позволяет строить значительно более сложные компоненты, адекватные элементам моделируемых физических систем. Однако, далеко не вся всех приложений является естественным представление модели в виде структурной схемы. В исследовательских задачах, в учебном процессе и на ранних стадиях разработки систем управления пользователь в основном имеет дело с моделями отдельных изолированных систем. Не случайно для этих задач чаще используются математические пакеты, ориентированные на изолированные системы и функциональный стиль описания, а не пакеты компонентного моделирования. Для моделирования изолированных систем актуальным является формирование сложного поведения из отдельных фрагментов. Эти модели далее становятся основой для разработки компонент, которые будут объединены в модель всей системы. Возможность пользователя разрабатывать укрупненные высокоуровневые компоненты, отражающие естественную структуру прикладной задачи, является одним из важнейших направлений развития современных инструментов моделирования. Представля-

ется, что в современном инструменте системно-аналитического моделирования структурные схемы должны использоваться только в том случае, когда они отражают естественную структуру системы и облегчают разработку и понимание модели. Формализм гибридного автомата позволяет строить сложные поведения путем последовательного, а не параллельного включения отдельных компонент. Очевидно, что динамические системы, приписываемые состояниям гибридного автомата, можно рассматривать как компоненты – экземпляры соответствующих классов. Таким образом, открывается своего рода «новое измерение» для ООМ. Используя возможность последовательного включения компонент, можно многие задачи моделирования решать вообще без составления структурных схем, ограничиваясь моделью изолированной системы.

Практически во всех существующих языках ООМ предполагается жесткая фиксация стереотипа атрибутов, который затем невозможно изменить в производном классе. Однако, опыт объектно-ориентированного программирования говорит о том, что использования наследования и полиморфизма объектов действительно чрезвычайно плодотворны, если вся иерархия классов тщательно продумана заранее. Однако, в ходе исследовательского моделировании сделать это практически невозможно. Прежде всего, это относится к определению стереотипа переменных, то есть разделения переменных на входы, выходы, контакты, параметры и т.д. На ранних этапах проектирования или научного исследования, на стадии отработки отдельных компонентов, чрезвычайно трудно предугадать возможные взаимодействия, которые могут потребоваться при объединении этих компонентов на последующих этапах. Например, если вы исследуете движение тела, брошенного под углом к горизонту, вполне естественно в качестве первого приближения использовать допущение о постоянстве ускорения силы тяжести и плотности воздуха, то есть задать эти величины как константы. Впоследствии вы можете установить, что рабочий диапазон дальностей требует уточнения модели и учесть зависимость ускорения силы тяжести и плотности от высоты. В руководстве

по языку Modelica именно этот пример демонстрируется как случай плодотворного использования inner/outer-функций для создания набора моделей различной точности. Однако, при декларации, например, ускорения силы тяжести как outer-функции вы, во-первых, сразу должны знать, что вам потребуются модели поля тяготения различной точности, а во-вторых, знать, что ускорение силы тяжести зависит только от высоты. В менее тривиальных случаях это требует специального исследования. Таким образом, язык системно-аналитического моделирования должен позволять изменять стереотип переменной в производном классе.

Глава 3. Математические модели сложной динамической системы.

В этой главе определяется набор базовых математических моделей непрерывной и гибридной системы, необходимых для создания языка и архитектуры инструментальных средств моделирования сложных динамических систем. Эти математические модели могут быть трех видов:

1) «внешние» модели, которые используются для описания отдельных компонентов и являются базовыми для языка моделирования;
2) «промежуточные» модели, которые возникают неявно при объединении компонентов в совокупную систему и должны обрабатываться инструментальными средствами;
3) «вычислимые» модели, которые непосредственно поддерживаются численными методами и методами имитационного моделирования, встроенными в инструментальные средства.

Инструментальные средства должны обеспечивать контроль корректности этих моделей и автоматическое преобразование «внешних» и «промежуточных» моделей к «вычислимым».

Математические модели непрерывной системы.

Непрерывная составляющая поведения является неотъемлемой частью описания любой гибридной системы. Поскольку для объектно-ориентированного анализа сложных технических систем не актуально глубокое исследование физических процессов, для описания непрерывных систем можно ограничиться обычными производными.

Математические модели непрерывной изолированной системы.

Будем использовать следующие обозначения:

- $t \in [t_0, \infty) \subset \Re$ - непрерывное время;

- $x = \{x_i \in \Re \mid i \in 1..n_x\}$ $x\{t_0) = x^0$ - вектор интегрируемых переменных (их также часто называют «дифференциальными» переменными);

- $y = \{y_i \in \Re \mid i \in 1..n_y\}$ - вектор искомых переменных в алгебраических уравнениях (их также часто называют «алгебраическими» переменными) с согласованными начальными значениями $y(t_0) = y^0$;

- $w = \{w_i \in \Re \mid i \in 1..n_v\}$ - вектор параметров.

Рассмотрим различные формы представления математической модели непрерывной системы, которые непосредственно, без дополнительных преобразований поддерживаются в существующих программных реализациях численных методов [59,81,72], то есть являются «вычислимыми» формами.

Форма 1.1. Система обыкновенных дифференциальных уравнений первого порядка, разрешенных относительно производных

$$\begin{cases} \dfrac{dx}{dt} = F(x, w, t)) \end{cases}$$

Предполагается, что функция F непрерывна по всем переменным в некоторой окрестности t_0. Известны также специальные численные методы, способные решать систему обыкновенных дифференциальных уравнений второго порядка, разрешенных относительно производных

$$\begin{cases} \dfrac{d^2x}{dt^2} = F(x, w, t)) \end{cases}$$

Форма 1.2. Система алгебраических уравнений

$$\{0 = G(y, w, t)$$

Предполагается, что функция G непрерывна по всем переменным в некоторой окрестности t_0 и система является корректной, то есть:

1) система является правильно определенной, то есть число алгебраических переменных равно числу уравнений;

2) система является структурно невырожденной, то есть, каждой алгебраической переменной можно сопоставить уравнение, из которого ее можно определить.

Заметим, что корректность системы уравнений вовсе не означает отсутствия проблем при ее численном решении.

Форма 1.1а. Обыкновенные дифференциальные уравнения первого порядка, неразрешенные относительно производных, но представимые в виде:

$$\begin{cases} A(x)\dfrac{dx}{dt} = F(x, w, t) \end{cases}$$

Предполагается, что матрица $A(x)$ является невырожденной.

Форма 1.1б. Обыкновенные дифференциальные уравнения первого порядка, неразрешенные относительно производных:

$$\begin{cases} G(\dfrac{dx}{dt}, x, w, t) = 0 \end{cases}$$

Предполагается, что система алгебраических уравнений относительно производных является корректной.

Подстановки (формулы). Часто в системе алгебраических уравнений можно выделить зависимости вида

$\{y_j = f_j(\bar{y}^j, t) \mid j = 1..n_y\}$, где $\bar{y}^j = \{y_i \mid i \in 1..n_y, i \neq j\}$,

то есть зависимости, выражение в правой части которых не зависит от переменной, стоящей в левой части. Такие зависимости рассматривают как подстановки (формулы), вводимые из-за удобства и не влияющие на структуру системы уравнений. При численном решении в целях повышения скорости решения их обычно рассматривают как простые присваивания. Будем полагать, что любая вычислительная форма предполагает также наличие подстановок. Набор формул должен быть правильно упорядочен, то есть значение переменной должно быть вычислено по соответствующей формуле прежде, чем оно будет использовано в правой части какой-либо другой формулы. Набор формул не должен содержать так называемых «алгебраических циклов», то есть косвенных зависимостей переменных от самих себя через другие формулы. При обнаружении «алгебраического цикла» он должен разрываться: одна из входящих в него формул должна рассматриваться как алгебраическое уравнение и решаться численно.

Форма 1.0. Возможен случай, когда непрерывная система задается только набором формул. Это соответствует найденному аналитическому решению системы алгебраических или дифференциальных уравнений, В этом случае для нахождения решения $y(t)$ вообще не нужны численные методы.

Форма 1.3. Система дифференциально-алгебраических уравнений, в которой дифференциальные уравнения первого порядка разрешены относительно производных

$$\begin{cases} \dfrac{dx}{dt} = F(x, y, w, t) \\ 0 = G(x, y, w, t) \end{cases}$$

Такая система называется системой дифференциально-алгебраических уравнений индекса 1 или системой обыкновенных дифференциальных уравнений с ограничениями. Их также называют системой дифференциально-алгебраических уравнений индекса 1 в форме Хессенберга. Теоретически система DAE индекса 1 путем дифференцирования алгебраического уравнения может быть сведена к системе дифференциальных уравнений. Однако, выполнить такое преобразование аналитически не всегда возможно и существует достаточное число апробированных численных методов, которые решают эту задачу в исходном виде [81,72].

Форма 1.4. Система дифференциально-алгебраических уравнений индекса 2 в форме Хессенберга

$$\begin{cases} \dfrac{dx}{dt} = F(x, y, w, t) \\ 0 = G(x, w, t) \end{cases}$$

Существуют немногочисленные и не очень апробированные численные методы, способные решать эту задачу в исходном виде [72].

Таким образом, системы уравнений вида 1.i могут рассматриваться как базовые «вычислимые» математические модели непрерывной изолированной системы. Очевидно, что эти формы неравноценны по требуемым вычислительным затратам: наиболее эффективно аналитическое решение в виде сис-

темы формул, затем следуют «чистые» алгебраические и дифференциальные уравнения и затем дифференциально-алгебраические уравнения (внутри двух последних групп также имеется упорядоченность по эффективности). Форма 1.3 является наиболее общей, но по возможности желательно преобразовывать исходное описание непрерывной системы к более эффективному (в идеале – получить аналитическое решение).

Однако, описание непрерывной системы в виде системы уравнений вида 1.i имеет слишком много ограничений и требует от пользователя много ручной работы по преобразованию уравнений. Более удобным является задание непрерывной системы в естественной форме (2). Эта форма включает в себя:

- использование дифференциально-алгебраических уравнений, не разрешенных относительно производных;
- использование производных более высоких порядков, чем первая.

$$(2)\quad F(\frac{d^k x}{dt^n},...,\frac{dx}{dt},x,y,w,t)=0\,,$$

В общем случае возможно, что начальные условия $x\{t_0)=x^0,\frac{dx}{dt}(t_0)=x'^0,...,\quad y(t_0)=y^0$ не являются согласованными). Необходимо найти решение $x(t),\quad y(t)$.

Надежных численных методов, непосредственно решающих задачу (2), в настоящее время не существует [72]. Поэтому система уравнений вида (2) может рассматриваться как базовая «внешняя» модель непрерывной системы только при условии, что имеется алгоритм преобразования ее к виду (1.i).

Компонентные модели непрерывных систем.

Компонентное моделирование предполагает построение модели непрерывной системы в целом из отдельных непрерывных компонентов.. Непрерывным компонентом C в общем случае будем полагать совокупность $C=\{v,v^0,Q\}$ набора переменных $v=(v_i\in\Re\,|\,i\in 1..m\}$, начальных значений v^0 и

системы уравнений Q в виде (1) или (2), где $x \subseteq v, y \subseteq v, w \subseteq v$. Отличием компонента от изолированной системы является то, что для компонента часто ничего нельзя сказать о корректности системы уравнений Q вне контекста ее взаимодействия с другими компонентами.

Результатом простого объединения непрерывных компонентов A и B является изолированная непрерывная система $S = A + B$, для которой $v_S = v_A \cup v_B$, а $Q_S = Q_A \cup Q_B$. Заметим, что здесь мы неявно пользуемся представлением о том, что непрерывное время t является независимым и глобальным.

Рассмотрим объединение взаимодействующих непрерывных компонентов. Традиционным способом описания такого объединения является функциональная схема. Функциональная схема строится из структурных компонентов, соединенных функциональными связями. Предполагается, что входящие в функциональную схему компоненты функционируют одновременно и параллельно. Функциональная схема эквивалентна изолированной непрерывной системе S, для которой $v_S = v_A \cup v_B$, $Q_S = Q_A \cup Q_B \cup L(A,B)$, где $L(A,B)$ - уравнения связей

Структурный компонент предполагает разбиение набора переменных v на две составляющих: $v = v^E \cup v^S$, где v^E - вектор внешних переменных, а v^S - вектор внутренних переменных (вектор состояния). Функциональные связи могут соединять только внешние переменные компонентов. В зависимости от ограничений, накладываемых на внешние переменные, различают направленные (ориентированные) и ненаправленные (неориентированные) структурные компоненты [4,34,108]. Направленный структурный компонент обычно называют блоком.

Направленный структурный компонент подразумевает представление в классическом формализме «вход-состояние-выход» [27], то есть внешние переменные разбиваются на входы и выходы $v^E = v^I \cup v^O$, а поведение задается отображением $(v^I, v^S) \to (v^S, v^O)$. Таким образом, значение входа не может

быть изменено внутри блока, а значение выхода, напротив, может быть изменено только внутри блока. Для непрерывных блоков это означает, что входами могут являться только элементы вектора параметров $v^I \subseteq w$, а дифференциальные и алгебраические переменные могут быть только состояниями или выходами: $v^O \subseteq x \cup y$. Направленные компоненты могут соединяться только направленными (ориентированными) связями, которые передают значение выхода одного компонента на вход другого (или того же самого) компонента (Рис. 9). Функциональная схема, построенная только с использованием направленных компонентов и связей, называется блок-схемой. Описание взаимодействий с помощью «входов» и «выходов» поэтому и называют «причинно-следственным» («casual») подходом в моделировании, поскольку в направленной связи только источник может влиять на приемники, но не наоборот. Уравнением направленной связи является формула

приемник = *источник* .

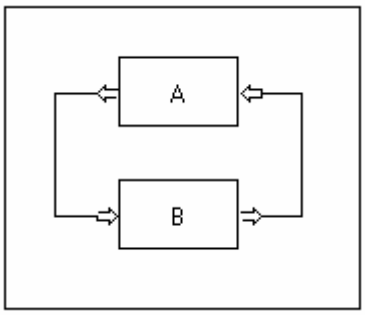


Рис. 9

Рассмотрим систему, построенную из направленных блоков A и B, соединенных направленными связями (Рис. 9). Уравнения связей для этой системы имеют вид простейших формул

$$\begin{cases} \hat{w}_A = \hat{z}_B \\ \hat{w}_B = \hat{z}_A \end{cases},$$

где $\hat{w}_A \subseteq w_A, \hat{w}_B \subseteq w_B, \hat{z}_A \subseteq x_A \cup y_A, \hat{z}_B = x_B \cup y_B$

Ясно, что эти формулы не влияют на структуру уравнений Q_A, Q_B и если поведение блоков представлено в форме 1.i, то система уравнений Q_S

также относится к виду 1.i и является корректной, если корректными являются системы уравнений Q_A и Q_B.

Таким образом, при использовании ориентированных блоков и связей можно проводить преобразование исходной системы уравнений к форме 1/I и анализ корректности преобразованной системы уравнений для каждого блока отдельно. При объединении ориентированных блоков необходимо лишь провести дополнительную сортировку формул, соответствующих направленным связям, а также обнаружить и устранить возможные алгебраические циклы, которые могут появиться из-за добавления формул связей.

В ненаправленном (неориентированном) структурном компоненте на внешние переменные не накладывается никаких ограничений: они могут изменяться как вовне блока, так и внутри блока, то есть $v^E \subseteq x \cup y \cup w$. Внешние переменные в этих блоках подразделяют на «контакты» и «потоки» $v^E = v^C \cup v^F$ по виду уравнений, приписываемым их связям [108]. Связь, соединяющая два контакта a и b эквивалентна алгебраическому уравнению $a - b = 0$. Связь, соединяющая два потока, эквивалентна алгебраическому уравнению $a + b = 0$ (аналог закона Кирхгофа) [2]. В обоих случаях уравнения связи – это линейные алгебраические уравнения. Будем полагать внешние переменные вида «поток» частным случаем контактов.

Оказывается, что поведение отдельного ненаправленного блока достаточно сложно сформулировать в форме 1. Это связано прежде всего с тем, что для отдельного блока, не зная его связей с другими блоками, часто невозможно сказать, какие переменные являются искомыми. Например, в поведении блока «Сопротивление», задаваемом законом Ома $U = I \cdot R$, неизвестными являются как напряжение U, так и ток I и система уравнений является недоопределенной. Даже в казалось бы очевидном случае, когда блок задает перемещение тела под действием силы и описывается уравнением $m \cdot \frac{dV}{dt} = F$, ниоткуда не следует, что именно скорость V является искомой величиной. Если переменная V является контактом и связана ненаправленной связью с

генератором синусоидального сигнала, то в результирующей системе уравнений $\begin{vmatrix} m \cdot \dfrac{dV}{dt} = F \\ V = A \cdot \sin \omega \cdot t \end{vmatrix}$ именно переменная F будет искомой, то есть будет решаться задача, какой должна быть зависимость $F(t)$, чтобы скорость тела менялась согласно заданной зависимости. Не случайно ненаправленные блоки ассоциируются с моделированием физических систем, поскольку физические законы не формулируются в форме «дано …, найти …». Поэтому анализ корректности и выбор искомых переменных в случае ненаправленных блоков возможен только для всей схемы в целом.

Рассмотрим систему, построенную из ненаправленных компонентов A и B, соединенных ненаправленными связями через контакты (Рис. 10).

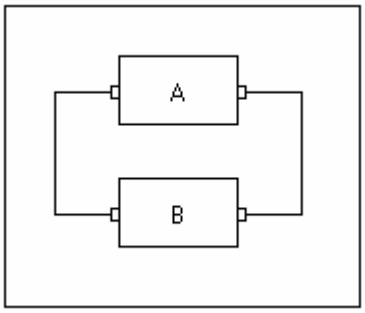


Рис. 10

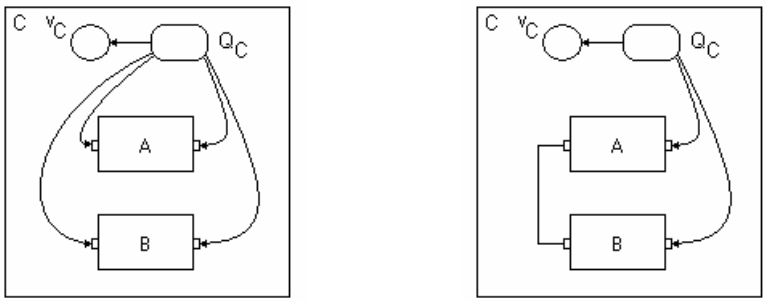
Уравнения связей будут иметь вид

$$\hat{v}_A^E - \hat{v}_B^E = 0,$$

где $\hat{v}_A{}^E \subseteq x_A \cup y_A \cup w_A$ и $\hat{v}_B{}^E \subseteq x_B \cup y_B \cup w_B$. Ясно, что даже, если системы уравнений Q_A и Q_B имеют вид (1), то результирующая система уравнений в общем случае будет иметь вид (2б), поскольку уравнения связей могут ввести алгебраические зависимости между дифференциальными переменными, а также совершенно изменить соответствие искомых переменных и уравнений. В случае, когда системы уравнений Q_A и Q_B имеют вид (2), результирующая система уравнений также будет иметь вид (2), то есть множество непрерывных систем вида (2) замкнуто относительно операции объединения.

Таким образом, при объединении ненаправленных компонентов анализ корректности уравнений и преобразование их к вычислимой форме можно проводить только для системы в целом.

Возможен и более сложный случай объединения непрерывных компонентов, когда компонент-контейнер C также имеет свой собственный набор переменных v_C и систему уравнений Q_C, в которой используются как собственные переменные v_C, так и внешние переменные компонентов A и B, то есть $x_C \cup y_C \cup w_C \subseteq v_A^E \cup v_B^E \cup v_C$ (Рис. 11а). Например, система уравнений Q_C может просто воспроизводить уравнения связей (Рис. 10).



а) б)

Рис. 11

В случае ориентированных компонентов A и B в уравнениях Q_C должны соблюдаться ограничения для входов и выходов, то есть $x_C \cup y_C \subseteq v_A^I \cup v_B^I \cup v_C$ и $w_C \subseteq v_A^O \cup v_B^O \cup v_C$. Блоки A и B могут одновременно взаимодействовать и через связи (Рис. 11б). Таким образом вид результирующей системы уравнений зависит от направленности блоков. В самом общем случае мы получим систему уравнений в форме (2).

Наконец, возможен случай локального непрерывного компонента $C^S = \{v \cup v^S, v^0, Q\}$, «погруженного» в компонент-контейнер S и оперирующего как со своими собственными переменными, так и с переменными компонента-контейнера.

Пустая непрерывная система.

Введем специальный случай непрерывного компонента – пустую непрерывную систему $NULL = \{v, v^0, \varnothing\}$. Пустая непрерывная система в общем случае может иметь переменные, но не имеет уравнений. Функционирование пустой непрерывной системы в непрерывном времени заключается в сохранении переменными своих начальных значений $v(t) = v^0$. Введем также полностью пустую непрерывную систему $null = \{\varnothing, \varnothing, \varnothing\}$. Очевидно, что для любой непрерывной системы C имеет место равенство $C + null = C$. Заметим, что модель, состоящая из одной только пустой (или даже полностью пустой) непрерывной системы отнюдь не бессмысленна: она продвигает непрерывное время t. Как будет показано ниже, пустая непрерывная система играет очень важную роль в гибридных моделях.

Преобразование описания непрерывной системы к вычислимой форме.

Ниже прелагается комплексный алгоритм преобразования системы уравнений в форме (2) к форме (1.i) с одновременной структурной оптимизацией.

Строго говоря, исходная система уравнений может быть задана пользователем в еще более свободной форме

$$(3)\quad Q:\quad \{F_j(\frac{d^k X}{dt^n}, ..., \frac{dX}{dt}, X, Y, t) = 0 \mid j \in 1..N_Q\}$$

На этой стадии анализа мы еще не знаем, какие переменные являются дифференциальными, какие алгебраическими и какие параметрами, X здесь обозначает лишь множество переменных, производные которых использованы в уравнениях, а Y - множество всех остальных переменных.

$X \cup Y = x \cup y \cup w$

Шаг 1.

На этом шаге исключим из анализируемой системы производные. Введем для каждой производной дополнительную алгебраическую переменную

$$z = \{\frac{dx}{dt}, \frac{d^2x}{dt^2}, ..., \frac{d^kx}{dt^n}\}$$

Тогда систему (2) можно представить в виде

$$(4)\ \begin{cases} der(x) = z_1 \\ der(z_1) = z_2 \\ \dots \\ der(z_{k-1}) = z_k \\ 0 = F(z, X, Y, t) \end{cases}$$

где $der()$ обозначает лишь операцию дифференцирования. Систему уравнений (4) будем далее рассматривать как чисто алгебраическую.

Шаг 2.

В системе (4) имеется $N_Q + k$ уравнений и $k + N_X + N_Y$ переменных. Система уравнений должна быть правильно определенной, то есть число уравнений должно быть равно числу переменных. Если система переопределена, то есть $N_Q > N_X + N_Y$, то она ошибочна и дальнейший анализ невозможен. На практике заданная пользователем система уравнений обычно оказывается недоопределенной, то есть $N_Q < N_X + N_Y$. Необходимо выделить множество переменных V и множество параметров W, таких, что $V \cup W = z \cup X \cup Y$. Это возможно только для изолированной системы или для направленной компоненты.

Появление недоопределенных систем уравнений связано с тем, что физические законы не формулируются в форме «дано …, найти …». Например, в системе уравнений движения

$$\begin{cases} m \cdot \frac{d^2x}{dt^2} = F - \mu \cdot \frac{dx}{dt} \\ x = \sin(t) \end{cases}$$

можно искать $\{x, F\}$, можно $\{x, m\}$, а можно и $\{x.\mu\}$.

Выделение искомых переменных проводится по следующим эвристическим правилам:

- константы, параметры компоненты и входные переменные компоненты переносим в множество W;

- все элементы множества z переносим в множество V;

- стараемся перенести в множество V оставшиеся элементы множества X и те оставшиеся переменные из множества Y, которые стоят в левых частях формул;

- если после этого система не стала правильно определенной, обращаемся за помощью к пользователю в диалоговом режиме (в описании ориентированных компонент пользователь должен иметь возможность также явно указать, какие переменные должны быть искомыми).

Шаг 3.

На этом шаге производится проверка системы уравнений на структурную вырожденность. Система алгебраических уравнений является структурно невырожденной, если можно найти взаимно однозначное отображение $S: Q \leftrightarrow V$, сопоставляющее каждому уравнению переменную, которая определяется из этого уравнения при решении системы методом подстановки [114]. Очевидно, что в общем случае для системы уравнений (4) существует несколько отображений S. Мы ставим задачу нахождения любого отображения, такого, что:

- оставшиеся элементы множества X по возможности будут дифференциальными переменными в результирующей системе уравнений;

- в уравнениях, явно заданных пользователем в виде формул, по возможности искомой переменной будет переменная, стоящая в левой части формулы.

Любое отображение S, удовлетворяющее этим условиям, будем называть структурно оптимальным.

Составим квадратную символьную матрицу D размерности $N_Q \cdot N_Q$, в которой элемент D_{ij} содержит:

- символ «+», если переменная V_j участвует в уравнении Q_i;

- символ «-», если переменная V_j не участвует в уравнении Q_i;

- символ «*», если переменная V_j является искомой в уравнении Q_i;

- символ «@», если переменная V_j является параметром в уравнении Q_i.

Уравнения пронумеруем следующим образом: сначала уравнения

$$\begin{cases} der(x) = z_1 \\ der(z_1) = z_2 \\ \dots \\ der(z_{k-1}) = z_k \end{cases},$$

полученные при замене производных на алгебраические уравнения, затем формулы, а затем оставшиеся алгебраические уравнения. Соответственно переменные V пронумеруем аналогично: сначала искомые переменные из множества X, затем переменные из множества z, затем переменные, стоящие в левых частях формул, а затем оставшиеся переменные.

В начале выполнения шага 3 матрица D содержит только символы «+» и «-». Далее для всех строк $i^* \in 1..N_Q$ производится назначение искомой переменной:

1) в строке i^* определяется очередной элемент $D[i^*, j^*]$ (по возрастанию j), содержащий символ «+»;

2) если такой элемент не находится, то производится возврат на строку $i^* - 1$, если же $i^* = 1$, то система уравнений является структурно вырожденной;

3) если такой элемент находится, то во всех элементах матрицы $\{D[i^*, j] \mid j \in j^* + 1..N_Q\}$ и $\{D[i, j^*] \mid i \in i^* + 1..N_Q\}$ символ «+» заменяется на символ «@»;

4) если $i^* = N_Q$, то анализ успешно закончен, иначе производится переход на строку $i^* + 1$;

5) при возврате на данную строку во всех элементах матрицы $\{D[i^*, j] \mid j \in j^* + 1..N_Q\}$ и $\{D[i, j^*] \mid i \in i^* + 1..N_Q\}$ символ «@» заменяется на символ «+» и далее выполняется пункт 1) данного алгоритма.

Таким образом, если отображение S найти не удалось (попытка назначения искомой самой правой переменной в самом верхнем уравнении не увенчалась успехом), то это означает, что система (4) является структурно вырожденной. Для более точной диагностики ошибки можно использовать более сложные и трудоемкие алгоритмы анализа (например, изложенные в работе [83]).

Если же отображение S найдено, то оно является структурно оптимальным, поскольку матрица D построена таким образом, что формульные переменные и переменные – аргументы функции дифференцирования не назначаются искомыми только в случае, если никакие иные варианты не приводят к успешному результату.

Шаг 4.

На этом шаге проверяется индекс получившейся системы дифференциально-алгебраических уравнений и в случае обнаружения проблемы высокого индекса предпринимаются необходимые действия.

Если одна или несколько переменных – аргументов функции дифференцирования не назначены искомыми в соответствующих уравнениях, то это означает, что результирующая система дифференциально-алгебраических уравнений имеет индекс выше 1 (индексом системы дифференциально-алгебраических уравнений называется число дифференцирований, которое необходимо, чтобы преобразовать эту систему к чистым дифференциальным уравнениям). В этом случае имеет место т.н. проблема высокого индекса. Это означает, что между переменными, претендующими на статус дифференциальных, имеется неявная алгебраическая зависимость [105].

В этом случае возможны следующие решения :

1) все оставляется как есть и в соответствующих уравнениях оператор дифференцирования заменяется на функцию численного дифференцирования;

2) для индекса, равного 2, можно использовать некоторые специальные численные методы [81];

3) проводится аналитическое дифференцирование уравнения, в котором дифференцируемая переменная является искомой.

Численное дифференцирование обычно вызывает проблемы при численном решении уравнений. В лучшем случае это влечет за собой использование более сложных и трудоемких методов, в худшем – невозможность найти решение.

Специальные численные методы для решения систем уравнений с высоким индексом в настоящее время являются недостаточно надежными и апробированными.

Символьное дифференцирование в принципе является чисто технической проблемой для любого выражения, заданного в аналитической форме. Проблемы возникают лишь при использовании в уравнениях функций, заданных пользователем.

При использовании аналитического дифференцирования уравнение

$F(\hat{V},t)=0$, где $\hat{V}=\{\hat{V}_i \mid i=1..m\}\in V$ - множество переменных, участвующих в данном уравнении, заменяется на уравнение

$$\sum_{i=1}^{m}\frac{\partial F}{\partial \hat{V}_i}\cdot\frac{d\hat{V}_i}{dt}$$

Если производные $\frac{dV_i}{dt}$ уже используются в системе уравнений, то анализ повторяется с шага 1. Если нет, то дифференцируются уравнения, в которых переменные $\hat{V}_i$ являются искомыми.

Шаг 5.

На этом шаге производится проверка совокупности формул на наличие алгебраических циклов, разрыв найденных циклов и сортировка формул.

Возможно также проведение оптимизации алгебраических уравнений. В определенных случаях, например, если выражения в алгебраическом уравнении представляют собой линейную комбинацию относительно искомой переменной, его можно аналитически разрешить относительно этой переменной, превратив в формулу. После анализа совокупной системы формул, возможно, некоторые из них будут снова преобразованы в алгебраические уравнения, но в этом случае число этих уравнений будет минимальным.

Шаг 6.

В полученной системе уравнений оставшиеся уравнения вида $der(A) = B$ заменяются на дифференциальные уравнения вида $\frac{dA}{dt} = B$.

Пример.

Проиллюстрируем работу приведенного алгоритма на конкретном примере. Рассмотрим электромеханическую систему, состоящую из электродвигателя постоянного тока, на валу которого жестко закреплен вентилятор (Рис. 12.).

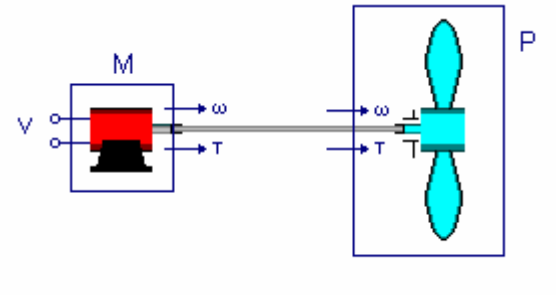


Рис. 12

Функционирование этой системы задается системой уравнений

$$(\text{П1})\begin{cases} Ja \cdot \dfrac{d\omega_M}{dt} = k_1 I + T_M \\ L_a \cdot \dfrac{dI}{dt} = V - k_2 \omega_M - R_a I \\ \dfrac{d\omega_P}{dt} = \dfrac{T_P - k_p \omega_P}{J_p} \\ T_M + T_P = 0 \\ \omega_M = \omega_P \end{cases}$$

На шаге 1 заменяем все производные на вспомогательные переменные и получаем систему уравнений

$$(\text{П2})\begin{cases}\dfrac{d\omega_M}{dt}=z_1 \quad (Q1)\\ \dfrac{dI}{dt}=z_2 \quad (Q2)\\ \dfrac{d\omega_P}{dt}=z_3 \quad (Q3)\\ J_a\cdot z_1=k_1 I+T_M \quad (Q4)\\ L_a\cdot z_2=V-k_2\omega_M-R_a I \quad (Q5)\\ z_3=\dfrac{T_P-k_p\omega_P}{J_p} \quad (Q6)\\ T_M+T_P=0 \quad (Q7)\\ \omega_M=\omega_P \quad (Q8)\end{cases}$$

На шаге 2 проводим правильное определение системы. Система (П2) является недоопределенной: на 8 уравнений имеется 18 переменных. Однако, величины $J_a, k_1, L_a, k_2, R_a, k_p, J_p$ являются параметрами системы, а переменная V является входной. Таким образом, остается 8 переменных: $X=\{\omega_m, I, \omega_p\}$, $z=\{z_1, z_2, z_3\}$, $Y=\{T_m, T_P\}$.

На шаге 3 строим первоначальную матрицу D:

	ω_M	I	ω_P	z_1	z_2	z_3	T_M	T_P
Q1	+	-	-	+	-	-	-	-
Q2	-	+	-	-	+	-	-	-
Q3	-	-	+	-	-	+	-	-
Q6	-	-	+	-	-	+	-	+
Q8	+	-	+	-	-	-	-	-
Q4	-	+	-	+	-	-	+	-
Q5	+	+	-	-	+	-	-	-
Q7	-	-	-	-	-	-	+	+

В результате применения алгоритма получаем матрицу

	ω_M	I	ω_P	z_1	z_2	z_3	T_M	T_P
Q1	*	-	-	@	-	-	-	-
Q2	-	*	-	-	@	-	-	-
Q3	-	-	@	-	-	*	-	-
Q6	-	-	@	-	-	@	-	*
Q8	@	-	*	-	-	-	-	-
Q4	-	@	-	*	-	-	@	-
Q5	@	@	-	-	*	-	-	-
Q7	-	-	-	-	-	-	*	@

Из найденного решения видно, что система уравнений имеет индекс 2 (решение уравнения $Q3$ требует дифференцирования переменной ω_P). Дифференцируем уравнение $Q8$, затем заменяем его на уравнение

$$\frac{d\omega_P}{dt} = \frac{d\omega_M}{dt}$$

и, повторив анализ сначала, получаем матрицу

	ω_M	I	ω_P	z_1	z_2	z_3	T_M	T_P
Q1	*	-	-	@	-	-	-	-
Q2	-	*	-	-	@	-	-	-
Q3	-	-	*	-	-	@	-	-
Q6	-	-	@	-	-	@	-	*
Q8	-	-	-	@	-	*	-	-
Q4	-	@	-	*	-	-	@	-
Q5	@	@	-	-	*	-	-	-
Q7	-	-	-	-	-	-	*	@

Эта система уравнений имеет индекс 1 и соответствует форме 1.3.

Наконец, разрешив аналитически алгебраические уравнения относительно искомых переменных и проведя анализ алгебраических циклов и сортировку формул, получаем итоговую систему уравнений

$$(\text{П3})\begin{cases}\dfrac{d\omega_m}{dt} = z_1\\ \dfrac{dI}{dt} = z_2\\ \dfrac{d\omega_P}{dt} = z_3\\ z_3 = z_1\\ T_P = z_3 \cdot J_P + k_p\omega_P\\ T_M = T_P\\ z_2 = \dfrac{V - k_2\omega_M - R_a I}{L_a}\\ J_a \cdot z_1 = k_1 I + T_M\end{cases}$$

В этой системе уравнений дифференциальными являются переменные $x = \{\omega_M, I, \omega_P\}$, алгебраическими $y = \{z_1\}$ и подстановками $s = \{z_1, z_2, T_P, T_M\}$.

Математические модели гибридного автомата.

В главе 1 было дано определение последовательного гибридного автомата. Однако, эта модель не обладает достаточной выразительностью, чтобы быть использована как базовая. Поэтому в качестве базовой внешней модели гибридной системы вводится обобщенный гибридный автомат. Далее в разделе рассматриваются вопросы параллельной композиции и алгоритмов интерпретации гибридных автоматов.

Последовательный гибридный автомат.

Уточним определение гибридного автомата, данное в главе 1 с учетом предложенной выше базовой модели непрерывной системы. Последовательным гибридным автоматом будем называть совокупность

$H = \{V, V^0, G\}$, где

- $V = \{V_C, V_D\}$ - множество переменных, включающее в себя множество непрерывных переменных $V_C = \{v_i \in \Re \mid i = 1..n_C\}$ $V_C = \{v_i \in \Re \mid i = 1..n_C\}$ и множество дискретных переменных $V_D = \{v_i \in \Re \cup I \cup Bool \cup Str \mid i = 1..n_D\}$, где I - множество целых чисел, $Bool = \{false, true\}$ - множество булевских значений, Str - множество строк.

- V^0 - множество начальных значений переменных;

- $G = \{g, B, P, A, F\}$ - граф переходов гибридного автомата.

Граф переходов включает в себя:

- $g = \{S, E, s^0\}$ - ориентированный граф, вершины которого сопоставлены элементам множества дискретных состояний автомата $S = \{s_i \mid i = 1..m_S\}$, одно из которых s^0 является начальным, а дуги сопоставлены возможным переходам автомата из одного состояния в другое $E = \{e_{ki} : s_i \to s_j \mid k \in 1..m_e, i \in 1..m_S, j \in 1..m_S\}$.

- $B = C^H \cup \{null\}$ - множество поведений, где $C^H = \{z_i^H \mid i = 1..m_Q\}$ - множество локальных непрерывных систем $z_i = \{V_i^Z \cup V, Q_i^Z, V_i^0\}$, где V_i^Z - множество вещественных переменных непрерывной системы, Q_i^Z - множество уравнений в форме (2), а V_i^0 - множество начальных условий для переменных V_i^Z. В общем случае $x_i^Z \subseteq V_C \cup V_i^Z$, $y_i^Z \subseteq V_C \cup V_i^Z$, $w_i^Z \subseteq V \cup V_i^Z$.

- $P = \{p_i(t, V) \in Bool \mid i = 1..m_P\} \cup \{true\}$ - множество логических предикатов.

- $A = \{a_i : V \to V_D \mid i = 1..m_A\}$ - множество мгновенных действий.

- $F = \{F_B, F_P, F_A\}$, где $F_B : S \to B$ - отображение, сопоставляющее множество систем уравнений множеству состояний (вершин графа), $F_P : E \to P$ - отображение, сопоставляющее множество предикатов множеству переходов (дуг графа), $F_A : E \to A$ - отображение, сопоставляющее множество мгновенных действий множеству переходов (дуг графа).

Очевидно, что в случае, когда $m_B = 0$, $F_B : S \to null$, последовательный гибридный автомат вырождается в чисто дискретную карту состояний.

Данное определение последовательного гибридного автомата неявно предполагает наличие глобального непрерывного времени $t \in T \subseteq [0, \infty]$.

Правила интерпретации последовательного гибридного автомата.

Фазовая траектория автомата H в пространстве состояний $S \times T$ представляет собой последовательный процесс [60], в котором дискретными событиями являются переходы из одного состояния в другое. На Рис. 13 показан пример такого процесса (закрашены состояния с ненулевой длительностью в непрерывном времени).

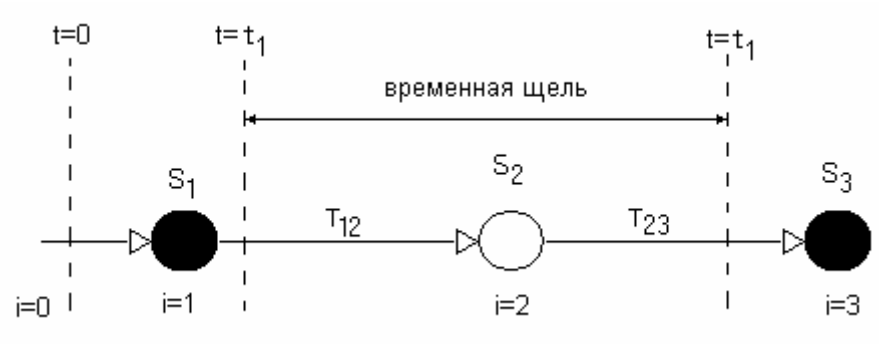


Рис. 13

В каждом из состояний $s \in S$ последовательный гибридный автомат ведет себя как непрерывная система $\{V_C, F_Q(s)\}$.

Рассмотрим подробнее фрагмент последовательного процесса, изображенный на Рис. 13.

На интервале $[t_0, t_1]$ автомат ведет себя как непрерывная система, поведение которой задается системой уравнений $F_Q(s_1)$. В момент t_1 предикат $F_P(T_{12})$ становится истинным, решение системы уравнений $F_Q(s_1)$ прекращается и значения переменных фиксируются как начальные $pre(V)$ в точке разрыва (.Рис. 14).

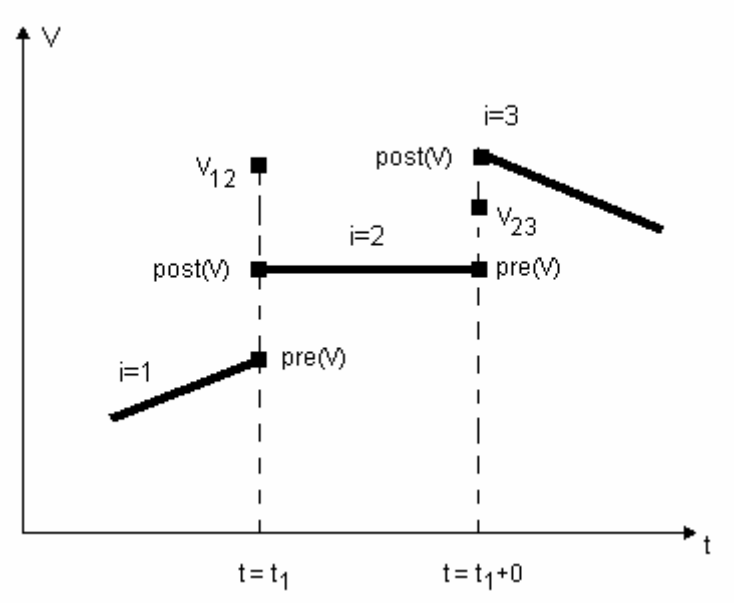


Рис. 14

Возможен случай, когда истинными становятся одновременно предикаты нескольких переходов, исходящих из текущего состояния. Предполагается, что существует какой-то критерий выбора из них одного (например, случайным образом, по указанию пользователя и т.п.).

Далее срабатывает переход T_{12} и выполняется последовательность мгновенных действий $F_A(T_{12})$. В результате выполнения этих действий значения некоторых переменных в общем случае изменяются и переменные приобретают значение $V_{12} \neq pre(V)$.

В результате срабатывания перехода состояние s_2 становится текущим и автомат начинает вести себя как непрерывная система, поведение которой задается системой уравнений $F_Q(s_2)$, для которой набор значений V_{12} является начальным. Однако, в общем случае этот набор значений является несогласованным и для его согласования необходимо решить алгебраическую составляющую системы уравнений $F_Q(s_2)$ и получить согласованные начальные значения $post(V)$ (Рис. 14). На этом действия в точке разрыва заканчиваются.

Однако, может оказаться, что для новых согласованных начальных условий становится истинным предикат какого-либо перехода из текущего со-

стояния (например, перехода T_{23})). В этом случае интервал существования непрерывной системы с системой уравнений $F_Q(s_2)$ равен нулю (дифференциальная составляющая системы уравнений не решается) и в точке $t_1 + 0$ начинается новый разрыв для которого значения переменных $post(V)$ для предыдущего разрыва становятся начальными значениями $pre(V)$ (Рис. 14). И так далее до состояния с ненулевой длительностью (на Рис. 13 это состояние s_3). Последовательность переходов и состояний с нулевой длительностью образует «временную щель» (Рис. 13).

Таким образом, если предположить, что исполняющая система пакета моделирования может реализовать эти правила интерпретации, то последовательный гибридный автомат можно считать «вычислимой» моделью изолированной гибридной системы. Ясно, что в этом случае все описания непрерывных систем также должны быть сведены к «вычислимой» форме.

Эквивалентность гибридного автомата и непрерывной системы.

Из введенных выше определений очевидно, что поведение непрерывной системы $C = \{V_C, Q_C\}$ совпадает с поведением последовательного гибридного автомата $H = \{V_H, G_H\}$, у которого $V_H = V_C$, $G_H = \{g, \{Q_C\}, \varnothing, \varnothing, F\}$, $g = \{\{s_0\}, \varnothing, s_0\}$, $F = \{\{s_0 \to Q_C\}, \varnothing, \varnothing\}$. То есть непрерывная система эквивалентна последовательному гибридному автомату с единственным состоянием без каких-либо мгновенных действий и переходов, которому приписана та же самая система уравнений (Рис. 15).

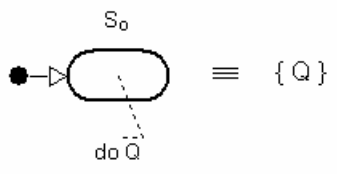


Рис. 15

Обобщенный гибридный автомат.

Рассмотренный выше последовательный гибридный автомат не обладает, однако, достаточными изобразительными возможностями, чтобы претендовать на роль базовой «внешней» модели гибридной системы.

Вернемся к рассмотренным в Главе 1 приемам моделирования систем с переменным поведением, которые используются в Simulink и Modelica:

- создаются специальные локальные компоненты, задаваемые блок-схемой или непосредственно описанием класса, соответствующие различным вариантам поведения;
- каким-то образом определяются дискретные события, соответствующие переключениям вариантов поведения;
- при обработке этих событий с помощью специальных блоков-коммутаторов или условных уравнений связи в работу включается нужный локальный компонент, а остальные исключаются из общей схемы и их функционирование приостанавливается.

Посмотрим теперь внимательно на определения гибридного автомата и машины (карты) состояний языка UML. Легко видеть, что оба эти определения по существу задают в визуальной форме описание последовательности функционирования в непрерывном времени отдельных составляющих сложного поведения объекта. Для гибридного автомата такой составляющей является локальная непрерывная система, а для карты состояний – непрерывная система *null* или локальная карта состояний (гиперсостояние). Составляющая поведения автоматически включается при входе в соответствующее состояние и отключается при выходе из него. Составляющие поведения взаимодействуют друг с другом через общие переменные активного объекта, значения которых становятся начальными условиями для следующей составляющей поведения.

Таким образом, гибридный автомат можно рассматривать как описание последовательной (во времени) композиции непрерывных компонентов, взаимодействующих через начальные условия. Карту состояний можно рас-

сматривать как описание последовательной композиции непрерывной системы *null* и локальных карт состояния, взаимодействующих через начальные условия.

В классических карте состояний и гибридном автомате локальный компонент, приписанный определенному состоянию, после выхода из этого состояния деактивируется, но продолжает существовать. Только при таком подходе можно осуществить прямой переход на внутреннее подсостояние или переход на т.н. «историческое» подсостояние, предусматриваемые в формализме машины состояний UML. Такую последовательную композицию компонентов мы будем называть *статической*.

Обобщим понятие последовательной (во времени) композиции компонентов по двум направлениям:

- будем рассматривать более общий случаи локального компонента, приписанного состоянию активного объекта ;
- будем рассматривать случай *динамической* последовательной композиции компонентов, при которой экземпляр локального компонента создается при входе в соответствующее состояние и уничтожается при выходе из этого состояния.

Неформально обобщенным гибридным автоматом будем называть последовательную композицию динамических или статических экземпляров локальных компонентов, которыми могут быть:

- непрерывная система (в том числе и *null*);
- обобщенный гибридный автомат;
- любая параллельная композиция непрерывных систем и обобщенных гибридных автоматов.

Предполагается, что последовательная композиция компонентов осуществляется по правилам построения машины состояний UML. Эти правила определяют момент смены компонента, следующий компонент, а также алгоритм вычисления начальных условий для нового компонента. Все остальные дета-

ли определения машины состояний UML касаются только удобства языковых конструкций.

Обобщенным гибридным автоматом будем называть совокупность

$H = \{V, V^0, G\}$, где

- $V = V_C \cup V_D$ - множество переменных, включающее в себя множество непрерывных переменных $V_C = \{v_i \in \Re \mid i = 1..n_C\}$ и множество дискретных переменных $V_D = \{v_i \in \Re \cup I \cup Bool \cup Str \mid i = 1..n_D\}$, где I - множество целых чисел, $Bool = \{false, true\}$ - множество булевских значений, Str - множество строк. Непрерывные переменные являются искомыми в локальных непрерывных компонентах, дискретные переменные изменяются в мгновенных действиях. В общем случае существует множество искомых переменных, которые изменяются и в мгновенных действиях $V_C' = V_C \cap V_D \neq \varnothing$;
- V^0 - множество начальных значений переменных;
- $G = \{g, B, P, A, F\}$ - граф переходов гибридного автомата.

Граф переходов включает в себя:

- $g = \{S, E, E^S, s^0\}$ - ориентированный граф, вершины которого сопоставлены элементам множества дискретных состояний автомата $S = \{s_i \mid i = 1..m_S\}$, одно из которых s^0 является начальным, а дуги сопоставлены возможным переходам автомата из одного состояния в другое $E = \{e_{ki} : s_i \to s_j \mid k \in 1..m_e, i \in 1..m_S, j \in 1..m_S\}$. Кроме того, существует некоторое множество «внутренних» переходов $E^S = \{e_i^S \mid i = 1..m_e^S\}$, которые выполняются в пределах одного и того же состояния;
- $B = \{z_i \mid i = 1..m_B\}$ - множество локальных компонентов-поведений, где локальный компонент может являться:
    - непрерывной системой $z_i = \{v_i \cup V, v_i^0 = f(t, V), Q_i\}$;
    - пустой непрерывной системой $z_i = null$;
    - обобщенным гибридным автоматом $z_i = \{v_i \cup V, v_i^0 = f(t, V), G_i\}$;
    - параллельной композицией компонентов $z_i = z_{i,1} + z_{i,2} + ... + z_{i,n_i}$.
- $P = \{p_i(t, V) \in Bool \mid i = 1..m_P\} \cup \{true\}$ - множество логических предикатов;

- $A = \{a_i : V \to V_D \mid i = 1..m_A\}$ - множество мгновенных действий $A = A^E \cup A^S_{en} \cup A^S_{ex}$, где

  - A^E - множество мгновенных действий переходов;
  - A^S_{en} - множество входных мгновенных действий;
  - A^S_{ex} - множество выходных мгновенных действий.

- $F = \{F_B, F_B^{'}, F^S_E, F^S_{en}, F^S_{ex}, F_P, F_A\}$, где:

1. $F_B : S \to B$ - отображение, сопоставляющее множество поведений множеству состояний (вершин графа),
   - $F_B^{'} : S \to Bool$ - отображение, сопоставляющее элементу множества состояний значение *true*, если этому состоянию приписан статический экземпляр локального компонента, и значение *false*, если этому состоянию приписан динамический экземпляр локального компонента,
   - $F^S_E : E^S \to S$ - отображение, сопоставляющее множеству внутренних переходов множество состояний;
   - $F^S_{en} : S \to A^S_{en}$ - отображение, сопоставляющее множество состояний множеству входных мгновенных действий:
   - $F^S_{ex} : S \to A^S_{ex}$ - отображение, сопоставляющее множество состояний множеству выходных мгновенных действий;
   - $F_P : E \cup E^S \to P$ - отображение, сопоставляющее множество переходов множеству предикатов,
   - $F_A : E \cup E^S \to A^E$ - отображение, сопоставляющее множество переходов множеству мгновенных действий переходов.

В определении обобщенного гибридного автомата неявно предполагает существование независимого глобального непрерывного времени $t \in T \subseteq [0, \infty]$.

Граф переходов G, для которого $F_B^{'} : S \to \{false\}$, то есть все локальные компоненты являются динамическими, будем называть ***картой поведений***, чтобы отличить этот случай от классической карты состояний, в которой все локальные компоненты являются статическими.

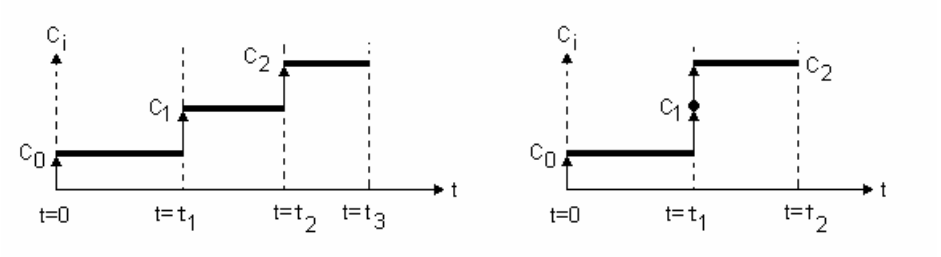
Следует отметить особое значение внутренних переходов для карты поведений. Для карты состояний внутренний переход эквивалентен внешнему, ведущему в то же самое состояние, если состояние не имеет входных и выходных действий. Для карты поведений эти переходы принципиально различны: внешний переход приведет к уничтожению экземпляра компонента, приписанного состоянию, и затем созданию нового экземпляра, в то время

как внутренний переход происходит при сохранении того же самого компонента.

Обобщенный гибридный автомат будем рассматривать как базовую «внешнюю» математическую модель гибридной системы. Везде далее под «гибридным автоматом» будем понимать обобщенный гибридный автомат.

Гибридное время.

Из определения гибридного автомата очевидно, что результатом его функционирования или функционирования любой композиции гибридных автоматов на интервале непрерывного времени T будет являться последовательность совокупных непрерывных систем $C_0, C_1, \ldots, C_i, \ldots, C_n$ (включая $NULL$), каждая из которых существует на определенном интервале $[t_i^0, t_i^e]$ (Рис. 16а).



а) б)

Рис. 16

В момент «склейки» двух последовательных непрерывных систем $t_i^e = t_{i+1}^0$ происходит совокупный мгновенный переход $C_i \to C_{i+1}$ (на Рис. 16 эти переходы показаны стрелками). Переменные гибридного автомата первоначально имеют значение $pre(V) = f_i(t_i^e)$. Затем согласно алгоритму определения новых начальных значений изменяются некоторые дискретные переменные из множества V_D и переменные гибридного автомата получают значение $post'(V)$. Однако, в общем случае эти начальные значения могут быть не согласованными и поэтому необходимо решить алгебраическую составляющую непрерывной системы C_{i+1}, чтобы согласовать начальные значения $post(V) = f_{i+1}(t_{i+1}^0)$.

Среди последовательных непрерывных систем возможны такие, длительность интервала существования которых равна 0 (например, система C_2 на Рис. 16б). Функционирование такой непрерывной системы ограничивается вычислением согласованных начальных значений. Мгновенную (в непрерывном времени) последовательность действий между двумя соседними интервалами ненулевой длительности называют «временной щелью».

Наличие временных щелей приводит к тому, что в общем случае зависимость переменных гибридного автомата от непрерывного времени $V(t)$ становится неоднозначной (Рис. 17).

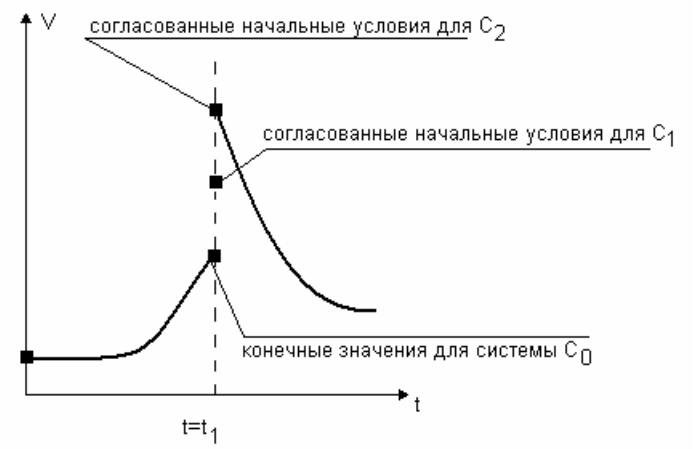


Рис. 17

Чтобы избежать этого, необходимо ввести глобальное гибридное время $h = (t,i) \mid t \in T \subset \Re, i \in \{0,1,2,...\}$, где t - непрерывная составляющая гибридного времени, равная значению непрерывного времени, а i - дискретная составляющая гибридного времени, равная номеру текущей совокупной непрерывной системы. Зависимость переменных гибридного автомата от гибридного времени $V(h)$ является однозначной (Рис. 18).

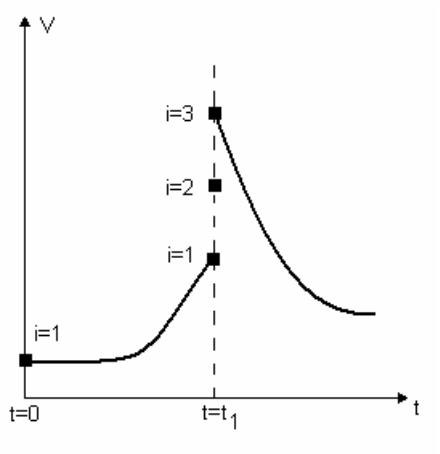


Рис. 18

Гибридный автомат может демонстрировать несколько случаев парадоксального поведения в гибридном времени.

Цикл во временной щели. При некоторых сочетаниях значений переменных последовательность совокупных непрерывных систем с нулевой длительностью во временной щели может оказаться циклической. В этом случае дискретное время будет неограниченно возрастать, а непрерывное время не будет изменяться.

Поведение Зенона. Предикаты, определяющие срабатывания переходов, могут быть заданы так, что существует предельное значение $t_f = \lim_{i \to \infty} t_i^e$. Например, если $t_i^e = t_i^0 + \Delta t$, а при каждом переходе выполняются мгновенные действия $\Delta t := \frac{\Delta t}{2}$, то при начальном значении $\Delta t = 1$ предельным значением непрерывного времени будет $t_f = 2$. В этом случае неограниченно возрастать будет дискретное время.

Эквивалентный последовательный гибридный автомат.

Последовательность непрерывных систем $C_0, C_1, \ldots, C_i, \ldots, C_n$ может быть порождена последовательным гибридным автоматом с состояниями $S_0, S_1, \ldots, S_n$, которым приписаны эти совокупные непрерывные системы, а также предикатами и мгновенными действиями переходов, обеспечивающими смену состояний в моменты t_i^e и вычисление нужных начальных значений.

Поскольку правила интерпретации последовательного гибридного автомата известны, то такой эквивалентный последовательный гибридный автомат может однозначно интерпретироваться исполняющей системой пакета моделирования и являться «внутренней» математической моделью гибридной системы.

Эквивалентный последовательный гибридный автомат в принципе может быть построен явно для заданного обобщенного гибридного автомата (если перевести конструкции языка Modelica в систему понятий гибридных автоматов, то именно так предписывает поступать спецификация этого языка). Такое явное описание будет достаточно громоздким даже для небольших значений n и, кроме того, делается невозможным моделирование систем с переменным составом. Однако, для воспроизведения работы эквивалентного автомата не особой необходимости в его полном явном описании (такая необходимость может возникнуть при анализе возможных траекторий автомата). Достаточно сформулировать конструктивный алгоритм определения последовательности мгновенных действия в следующем совокупном переходе $C_i \to C_{i+1}$ и определения следующей совокупной непрерывной системы C_{i+1} для любого момента i гибридного времени.

Обобщенный гибридный автомат отличается от последовательного гибридного автомата тем, что, во-первых, состоянию может быть приписан другой гибридный автомат и, во-вторых, состоянию может быть приписана параллельная композиция непрерывных систем и/или гибридных автоматов. Модель в целом также может быть представлена параллельной композицией непрерывных систем и/или гибридных автоматов. Ранее было показано, что любая параллельная композиция непрерывных систем дает в результате также непрерывную систему. Из определения обобщенного гибридного автомата видно, что результатом параллельной композиции гибридного автомата H и непрерывной системы C является гибридный автомат $\hat{H}$, для которого $\hat{V} = V^H \cup V^C$ и $\hat{B} = \{z_i^H + C \mid i = 1..m_B\}$ (то есть непрерывная система просто добавляется к каждому локальному поведению гибридного автомата).

Таким образом, нам нужно уметь динамически строить эквивалентный последовательный автомат для иерархического гибридного автомата и для параллельной композиции гибридных автоматов.

Иерархический гибридный автомат.

Иерархическим гибридным автоматом является такой гибридный автомат, в котором локальными компонентами могут быть непрерывная система или гибридный автомат. Рассмотрим самый нижний – терминальный - уровень вложенности иерархического автомата (Рис. 19).

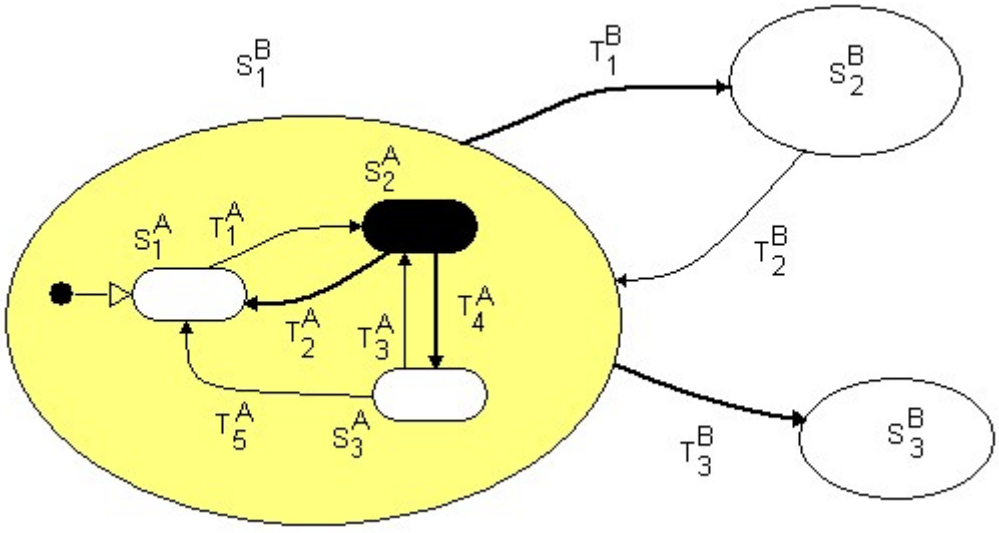


Рис. 19

Ясно, что на терминальном уровне гиперсостоянию может быть приписан только последовательный гибридный автомат. Поэтому текущей совокупной непрерывной системой будет являться непрерывная система, приписанная текущему состоянию этого автомата. На Рис. 19 показан последовательный автомат A, приписанный гиперсостоянию S_1^B иерархического автомата B. Полным текущим состоянием автомата B является состояние $S_1^B.S_2^A$ и текущей непрерывной системой является $F_B(S_2^A)$. Таким образом, если в иерархическом автомате текущим является гиперсостояние, то используются правила интерпретации для последовательного гибридного автомата терминального уровня. Единственным отличием является то, что в случае, если к срабатыва-

нию готовы одновременно переходы различных уровней вложенности, срабатывает только переход самого верхнего уровня (автоматы всех остальных уровней уничтожаются).

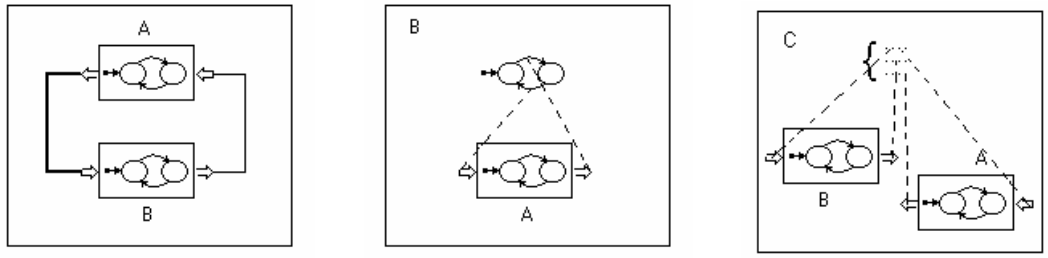
Принцип синхронной композиции гибридных автоматов.

Гибридные автоматы как структурные компоненты могут быть объединены:

а) путем создания функциональной схемы с явным указанием связей между их внешними переменными (Рис. 20а);

б) путем «погружения» в гибридный компонент-контейнер (Рис. 20б);

в) путем «погружения» в непрерывный компонент-контейнер (Рис. 20в).

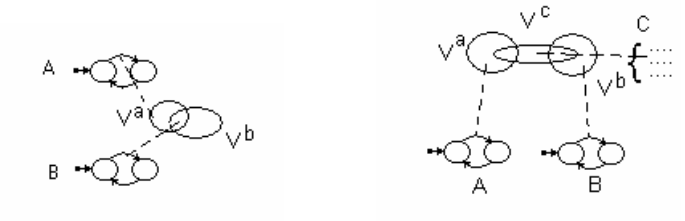


а) б) в)

Рис. 20

Для случая (а) существенным также является вид связываемых внешних переменных: 1) переменные являются чисто непрерывными, то есть принадлежат к множеству $V_C - V_C'$ (на Рис. 20а такая связь показана толстой линией); 2) переменные являются дискретными, то есть принадлежат множеству V_D (на Рис. 20а такая связь показана тонкой линией). Предположим, что связь между дискретными переменными является мгновенной дискретной, то есть при присваивании нового значения одной связываемой переменной немедленно изменяется значение второй переменной. Тогда случаи объединения гибридных автоматов (а2) и (б) будут эквиваленты совместной работе нескольких параллельных карт поведений над общими переменными (Рис. 21а),

а случаи (а1) и (в) – совместной работе параллельных карт поведений, взаимодействующих только через систему уравнений (Рис. 21б).



а) б)

Рис. 21

В первом случае мы получаем совокупность вида $\hat{H} = \{\hat{V}, \hat{G}\}$, где $\hat{G} = \{G_i \mid i = 1..n\}$ - множество графов переходов, а $\hat{V} = V_1 \cup ... \cup V_n$ - совокупное множество переменных, причем $V_1 \cap ... \cap V_n \neq \varnothing$. Во втором случае мы получаем совокупность вида $\hat{H} = \{\hat{V}, \hat{G}, L\}$, где $\hat{G} = \{G_i \mid i = 1..n\}$ - множество графов переходов, $\hat{V} = V_1 \cup ... \cup V_n \cup V^L$ - совокупное множество переменных, причем $V_1 \cap ... \cap V_n = \varnothing$ и $V_1 \cap ... \cap V_n \cap V^L \neq \varnothing$, а $L = \{V^L, Q^L\}$ - некоторая непрерывная система.

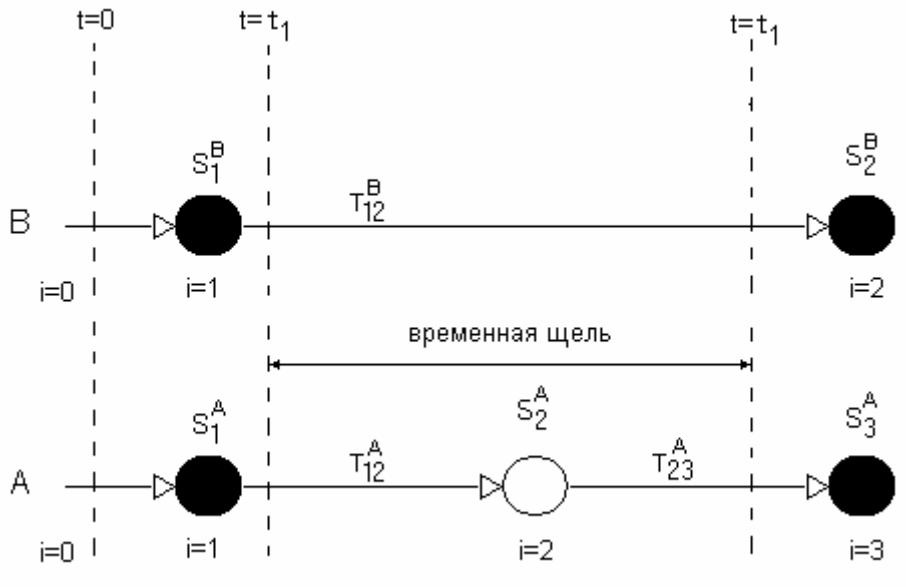


Рис. 22

Отличием гибридных автоматов от других дискретных моделей [12,60] является то, что они взаимодействуют через внешние переменные, а не через посылку сообщений. Дискретные события в параллельных гибридных авто-

матах естественно упорядочены по непрерывной составляющей гибридного времени автомата. Однако, дискретные составляющие гибридного времени отдельных автоматов несопоставимы. Поэтому в случае, когда в нескольких гибридных автоматах одновременно в непрерывном времени готовы к срабатыванию переходы, возникает проблема их синхронизации в гибридном времени. Пусть в гибридных автоматах A и B в момент t_1 одновременно готовы к срабатыванию переходы T_{12}^A и T_{12}^B (Рис. 22). Возможны три варианта выполнения этих переходов

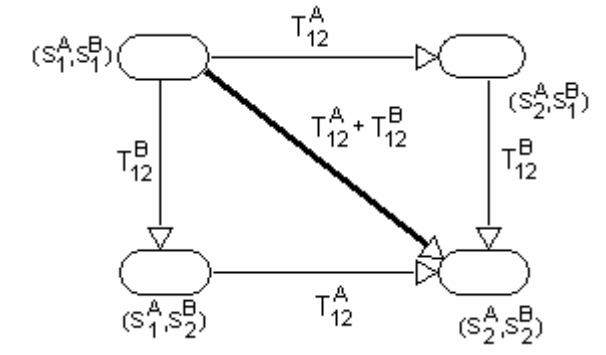


Рис. 23

Очевидно, что при последовательном выполнении переходов результат в общем случае будет зависеть от порядка выполнения и при наличии общих переменных, и при взаимодействии через непрерывную систему (в промежуточных состояниях будут согласовываться начальные условия). При выполнении объединенного перехода $T_{12}^A + T_{12}^B$ результат будет однозначным, если автоматы не имеют общих переменных. В результате выполнения объединенного перехода дискретная составляющая гибридного времени модели увеличится на 1, что соответствует интуитивному представлению о непрерывной одновременности..

Принцип синхронной композиции гибридных автоматов предполагает, что: 1) гибридные автоматы взаимодействуют только через непрерывные связи; 2) все готовые к срабатыванию переходы выполняются одновременно в гибридном времени модели. Отметим, что принцип синхронной комозиции не отменяет, а дополняет явную синхронизацию процессов через сообщения

(см. раздел 0). Результат синхронной композиции гибридных автоматов будем называть **синхронным параллельным автоматом**. Легко видеть, что синхронное объединение синхронного автомата с непрерывной системой, последовательным автоматом или синхронным автоматом в результате дает синхронный автомат. таким образом, синхронный параллельный автомат и является «промежуточной» математической моделью гибридной системы.

В языке Modelica аналогичная проблема решается с помощью принципа синхронного потока данных, в соответствии с которым все формулы в обработчиках дискретных событий добавляются к последней совокупной системе уравнений, включающей уравнения связей, и ищется решение этой «мгновенной» совокупной системы уравнений. Однако, мгновенные действия в автоматных моделях часто бывает затруднительно выразить в форме уравнений. Кроме того, «мгновенная» совокупная система алгебраических уравнений может оказаться вырожденной.

В примере 5 Приложения 1 показан случай, когда асинхронный гибридный автомат показывает качественно неверное поведение моделируемой системы, принцип синхронного потока данных приводит в вырожденной системе уравнений, в то время как модель, основанная на принципе синхронной композиции, дает верные результаты (см. также [37,100]).

Поэтому в данной работе предлагается использовать в языке системно-аналитического моделирования только синхронную параллельную композицию гибридных автоматов. Однако, для этого необходимо свести случаи объединения (а2) и (б), показанные на Рис. 20 к синхронному объединению. В случае (а2) для связей, в которых участвуют дискретные вещественные переменные, просто добавим соответствующие алгебраические уравнения к непрерывной системе L на общих основаниях. Даже если эти переменные чисто дискретные, то есть относятся к множеству $V_D - V_C$, то эти уравнения будут необходимы при нахождении согласованных начальных условий. Для направленных связей, в которых участвуют невещественные переменные (целые, булевские, строковые), добавим к системе L специальный набор

формул L_D, который не может участвовать в преобразованиях системы уравнений связи к вычислимому виду. Этот набор формул просто должен дополнительно вычисляться при каждом решении совокупной непрерывной системы. Ненаправленные связи с невещественными переменными запретим. В случае (б) необходимо ввести специальные «отражения» внешних переменных локального компонента в компоненте-контейнере и установить связи между внешними переменными и этими «отражениями» (Рис. 24). После этого данный случай сводится к предыдущему.

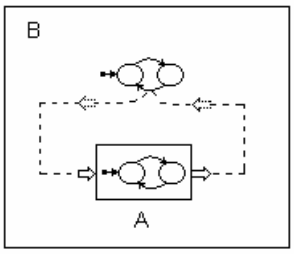


Рис. 24

Правила интерпретации синхронного параллельного гибридного автомата.

Параллельный гибридный автомат порождает n последовательных процессов, выполняющихся параллельно. На Рис. 22 показаны процессы, соответствующие объединению двух последовательных автоматов A и B.

Текущим состоянием эквивалентного последовательного автомата является набор $\hat{s} = (s^1,...,s^i,...,s^n)$, где s^i - текущее состояние i-й карты поведения на последнем уровне вложенности. Этому состоянию соответствует совокупная непрерывная система $\{\hat{V}, F_B^1(s^1) + F_B^2(s^2) + ... + F_B^n(s^n) + L)$. Например, для параллельного автомата, показанного на Рис. 22, начальным состоянием эквивалентного автомата является состояние $\hat{s}_1 = (s_1^A, s_1^B)$ и соответственно совокупной непрерывной системой $F_B(s_1^A) + F_B(s_1^B) + L$ (Рис. 25).

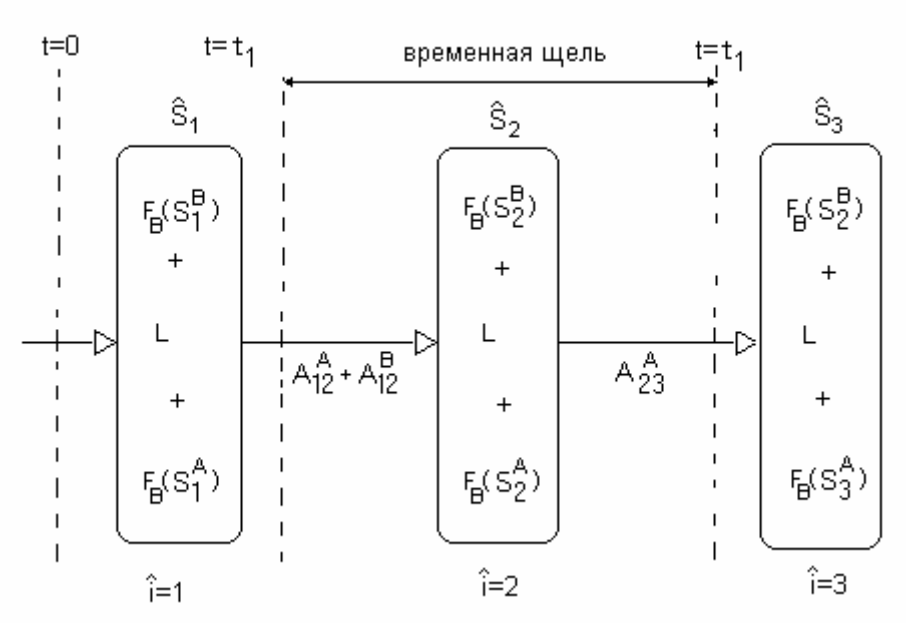


Рис. 25

Пусть $TO(s^i)$ - множество исходящих переходов для текущего состояния i-го последовательного автомата (с учетом гиперсостояний). Ближайшее дискретное событие происходит в момент t^*, когда становится истинным один или несколько предикатов из множества $\{F_P^i(E_j^i) | E_j^i \in TO(s^i), i = 1..n\}$. Функционирование текущей совокупной непрерывной системы прекращается, фиксируются значения $pre(\hat{V})$ и определяется множество срабатывающих переходов TE (предполагается, что для каждой последовательной карты поведений с учетом иерархии определяется единственный срабатывающий переход). Например, для автомата, показанного на Рис. 22, в момент $t = t_1$ $TE = \{T_{12}^A, T_{12}^B\}$. Следующим состоянием эквивалентного автомата $\hat{s}^*$ будет комбинация новых текущих состояний последовательных автоматов после срабатывания переходов, входящих в множество TE. Для примера на Рис. 22 это будет состояние $\hat{s}_2 = (s_2^A, s_2^B)$.

Переход эквивалентного автомата между этими состояниями будет являться эквивалентом множества переходов TE. Действия, выполняемые в этом переходе, будут являться объединением действий переходов, входящих в TE (включая выходные и входные действия в состояниях). Для примера на Рис. 22 $\hat{A}_{12} = A_{12}^A + A_{12}^B = A_{12}^B + A_{12}^A$. Для синхронного параллельного автомата опе-

рация объединения действий коммутативна, поскольку во время выполнения эквивалентного перехода уравнения связи не решаются. Для асинхронного параллельного автомата эта операция в общем случае не является коммутативной.

Далее определяется совокупная система уравнений и набор искомых непрерывных переменных для нового состояния эквивалентного автомата и для этой системы уравнений находятся согласованные начальные условия. В нашем примере это будет $F_B^A(s_2^A) + F_B^B(s_2^B) + L$ (Рис. 25). В общем случае совокупная система уравнений может соответствовать некоторой «промежуточной» форме. Предполагается, что исполняющая система пакета моделирования способна преобразовывать ее к «вычислимой» форме. Новое состояние становится текущим и все действия повторяются. В нашем примере состояние $\hat{s}_2 = (s_2^A, s_2^B)$ будет иметь нулевую длительность, поскольку после согласования начальных условий истинным становится предикат для перехода T_{23}^A. В результате эквивалентный автомат немедленно перейдет в состояние $\hat{s}_3 = (s_3^A, s_2^B)$.

Явная синхронизация гибридных автоматов с помощью сигналов.

Часто логика дискретного поведения моделируемой системы требует явной синхронизации двух или более последовательных процессов. Для этого обычно используется механизм сигналов [12]. Сигнал – это переменная специального типа `signal`. Послать сигнал можно только с помощью специального оператора `send`. Переменная-сигнал может фигурировать в качестве условия срабатывания перехода в других картах поведений. Пусть, например, в действиях перехода T_{12}^A посылается сигнал, по которому срабатывает переход T_{12}^C (Рис. 26). Рассмотрим, как соотносится явная синхронизация посредством сигналов с синхронной композицией автоматов.

Информация о посылке сигнала должна передаваться немедленно, не дожидаясь решения уравнений связи. Все переходы, ожидающие этого сиг-

нала, должны быть добавлены в множество TE (если их там еще нет) и выполнены в том же эквивалентном переходе. Например, для примера на Рис. 26 будет получен следующий эквивалентный процесс:

$$(s_1^A, s_1^B, s_1^C) \to [T_{12}^A + T_{12}^B + T_{12}^C] \to (s_2^A, s_2^B, s_2^C) \to [T_{23}^A] \to (s_3^A, s_2^B, s_2^C)$$

Таким образом, явная синхронизация параллельных гибридных автоматов с помощью сигналов приводит к динамическому формированию перехода в эквивалентном автомате во время его выполнения.

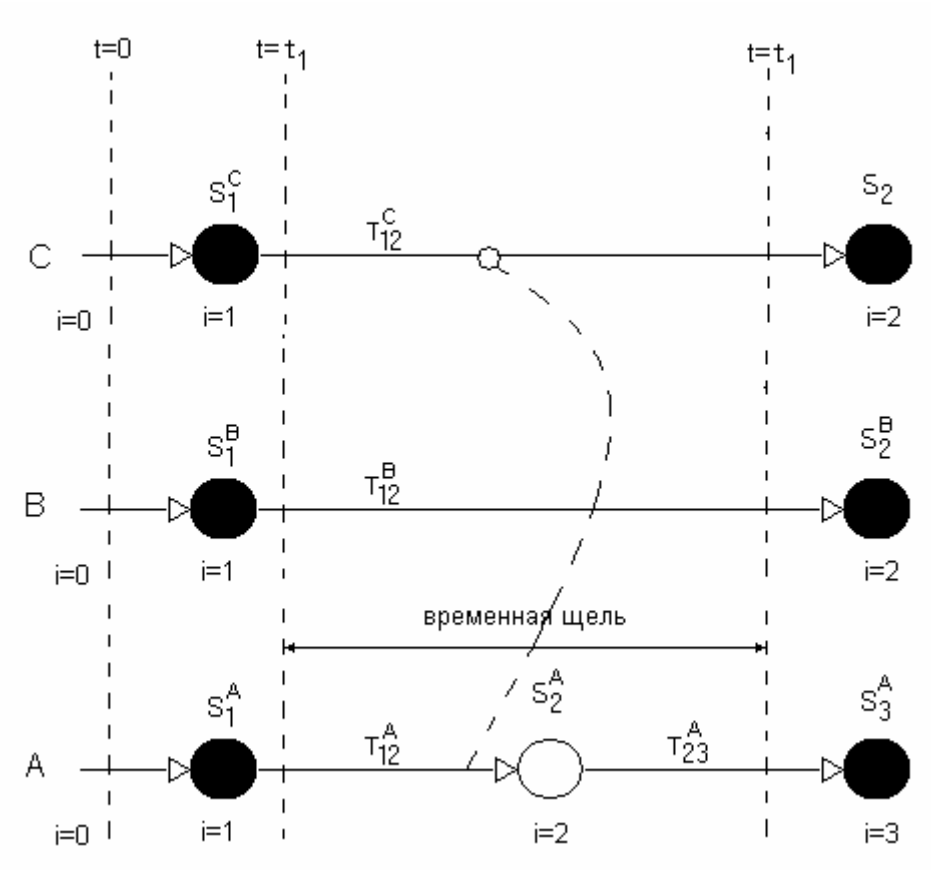


Рис. 26

Глава 4. Язык объектно-ориентированного моделирования сложных динамических систем.

В данной главе описываются принципы построения языка объектно-ориентированного моделирования сложных динамических систем MVL на базе математической модели обобщенного гибридного автомата.

Объекты и классы.

Объектом в языке MVL называется совокупность переменных и поведения, соответствующая математической модели обобщенного гибридного автомата. С точки зрения языка UML это активный объект, так как ему присуща своя собственная внутренняя деятельность, независимая от поведения других объектов. Однако, в отличие от программных объектов, эта деятельность может быть непрерывной, то есть не связанной ни с каким «потоком управления». Единственным «движителем» непрерывной деятельности является независимый и глобальный поток непрерывного времени. Кроме того, взаимодействие активных объектов, в отличие от UML, осуществляется не посредством вызова методов и передачи сообщений, а непрерывно через внешние переменные. Поэтому объект MVL, соответствующий модели обобщенного гибридного автомата, будем называть ***активным динамическим объектом***.

Рис. 27

Каждый объект является экземпляром соответствующего класса. Внешние переменные являются видимой частью объекта (интерфейсом), а поведение инкапсулировано внутри объекта (Рис. 27). Таким образом, разра-

ботчик класса может вносить изменения в определение поведения и внутренние переменные, не рискуя нарушить корректность любых случаев использования этого класса.

Описание поведения объекта в общем случае включает в себя следующие составляющие (Рис. 28):

- определения алгоритмических процедур и функций;
- определения локальных классов;
- структурную схему;
- систему уравнений:
- карту поведений (только одну, так как наличие нескольких параллельных карт поведений, оперирующих с одним и тем же набором переменных, противоречит принципу синхронного объединения гибридных автоматов).

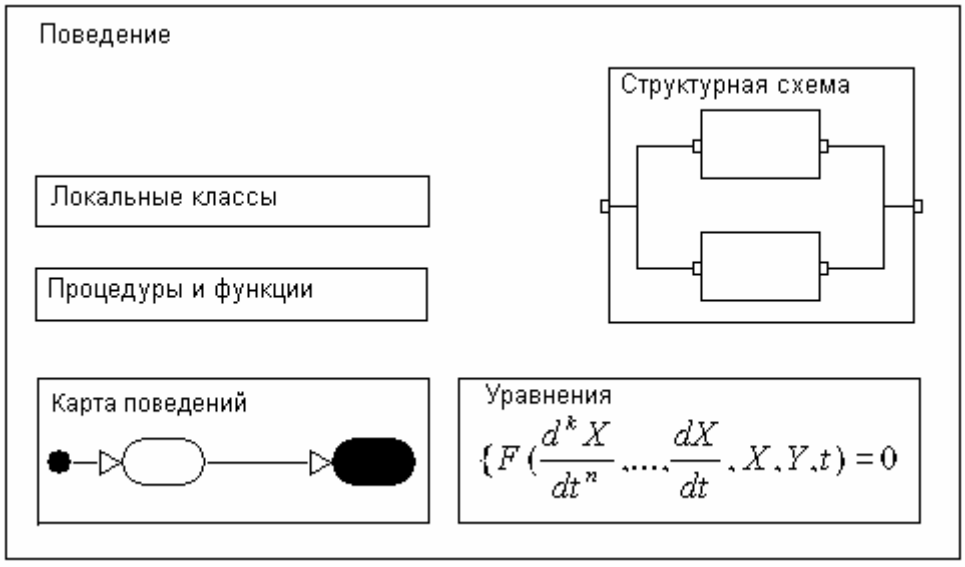


Рис. 28

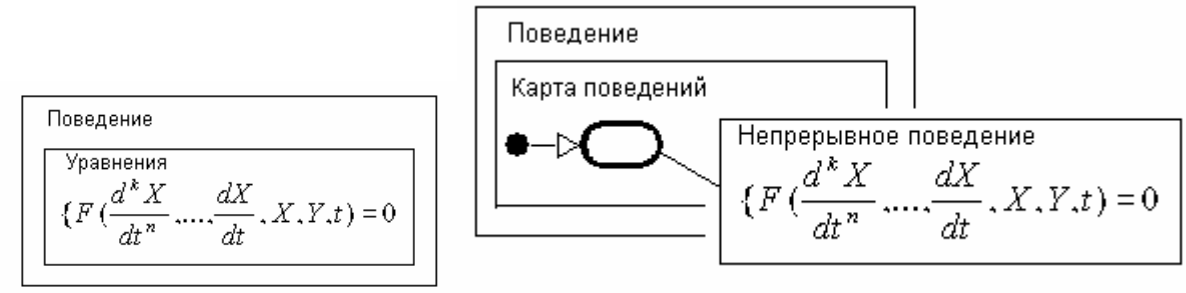
Таким образом, имеется возможность задать поведение активного динамического объекта с помощью любой комбинации этих составляющих. Все эти комбинации, как было показано в главе 2, сводятся к «внутренней» модели синхронного параллельного автомата, поддерживаемой исполняющей системой пакета моделирования.

Разделение описания класса на переменные и поведение отражает специфическую семантику активного динамического объекта. С точки зрения определения активного объекта UML переменные и локальные компоненты (элементы структурной схемы) являются атрибутами объекта, а процедуры и функции, гибридная карта состояний, уравнения, а также связи в структурной схеме являются элементами описания функционирования объекта. Класс имеет собственную область видимости имен, все элементы описания класса должны иметь несовпадающие имена.

Поведение активного динамического объекта в целом является результатом объединения всех заданных элементов поведения. Различные специальные сочетания этих элементов позволяют в качестве частных случаев получить важнейшие типовые объекты:

- непрерывный элементарный компонент:
- дискретный элементарный компонент:
- гибридный элементарный компонент:
- компонент-контейнер.

Непрерывный элементарный компонент соответствует классу, в определении поведения которого присутствует только система уравнений (Рис. 29а).



а) б)

Рис. 29

В определении непрерывного компонента могут также присутствовать определения внутренних переменных и алгоритмических функций, используемых в выражениях.

Рис. 30

Дискретный элементарный компонент соответствует классу, в определении поведения которого присутствует только карта поведений, у которой на последнем уровне вложенности всем состояниям приписано поведение *null* (Рис. 30). Отличия такой вырожденной карты поведений, которую далее будем называть дискретной, от иерархической карты состояний UML будут рассмотрены ниже.

Гибридный элементарный компонент соответствует классу, в определении поведения которого присутствует только карта поведений (Рис. 31).

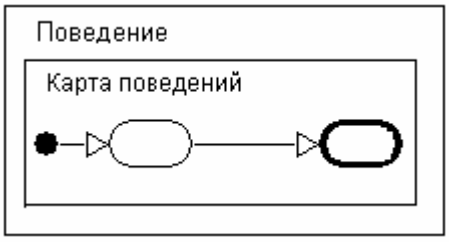
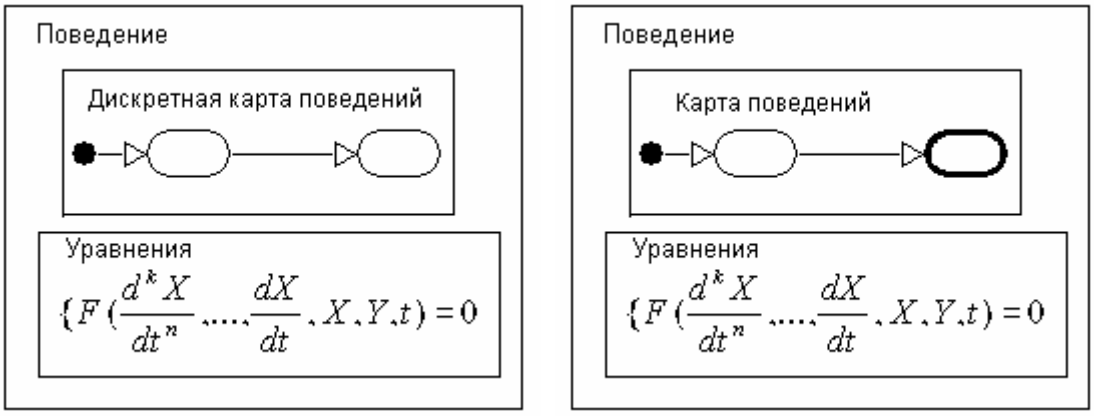


Рис. 31

Из общего определения объекта MVL можно также получить также другие сочетания карты поведений и уравнений, показанные на Рис. 32, которые в результате дают карту поведений.

В частности, объединение дискретной карты поведений и системы уравнений (Рис. 32а) соответствует гибридной модели, построенной с использованием подсистемы STATEFLOW и непрерывных блоков SIMULINK. Однако, приемлемость этих сочетаний для языка системно-аналитического моделирования является спорной. С одной стороны, такая конструкция позволяет достаточно удобно описывать гибридные модели, в которых набор переменных и уравнения не меняются, а лишь значения некоторых переменных изменяются

скачками (например, модель прыгающего мячика в Приложении 1). С другой стороны, опыт эксплуатации пакета Model Vision 2.1 показывает, что пользователи достаточно часто не в состоянии правильно оценить в уме результаты совместного функционирования карты поведений и «фоновой» системы уравнений, что приводит к ошибкам. Поэтому в языке MVL допускается только сочетание, показанное на Рис. 32а. Для случая, показанного на Рис. 32б, когда в системах уравнений, приписываемых различным состояниям, удобно выделить некоторую постоянную составляющую, можно использовать непосредственно операцию композиции систем уравнений (см. пример 4 Приложения 1 и Рис. 96).



а) б)

Рис. 32

На Рис. 29б показан частный случай гибридного компонента, поведение которого полностью эквивалентно непрерывному компоненту на Рис. 29а и который в принципе может использоваться вместо непрерывного компонента. Однако, опыт использования такого компонента в пакете Model Vision 3.x показал, что большинство пользователей воспринимает его как некоторую искусственную сущность, навязываемую в угоду совместимости с UML. Поэтому этот случай в пакете Model Vision 4.x рассматривается как переходный при преобразовании объекта из непрерывного в гибридный и наоборот.

Компонент-контейнер соответствует классу, в определении поведения которого присутствует только структурная схема и взаимодействие локальных объектов осуществляется явно через функциональные связи (Рис. 33).

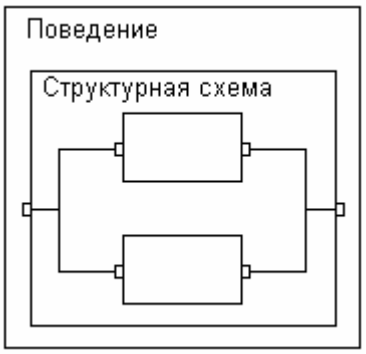
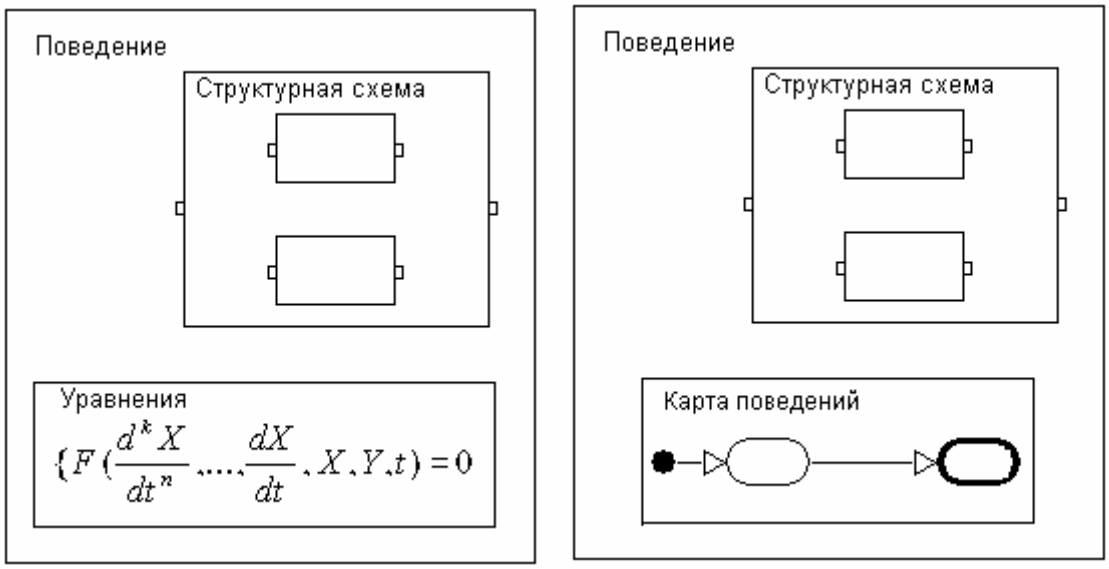


Рис. 33

Возможен также компонент-контейнер с собственным поведением, в котором взаимодействие локальных объектов осуществляется неявно через уравнения или карту поведений контейнера (Рис. 34).



а) б)

Рис. 34

Алгоритмические функции и процедуры используются для задания типовых вычислений или последовательностей действий в соответствии с принципами структурного программирования. Следует отметить, что исполь-

зование алгоритмических функций в системах уравнений может вызвать трудности при необходимости символьного дифференцирования уравнений.

Локальные классы используются для определения объектов, приписанных состояниям карты поведений. В отличие от внешних классов в определении локального класса видимы все переменные охватывающего класса.

Создание и уничтожение активного динамического объекта осуществляется:

- явно с помощью операторов `new` и `delete`, выполняемых в последовательности мгновенных дискретных действий;
- неявно при создании и уничтожении объекта-контейнера, в состав которого данный объект входит в качестве локального;
- неявно при входе в состояние и выходе из состояния, которому приписан данный объект.

Экземпляр класса может быть именованным, и анонимным. Объект может быть также определен неявно как композиция других объектов.

Пакеты и проект.

Пакет — это контейнер для группы компонентов, ограничивающий область их видимости. Компоненты, объявленные как экспортируемые, видимы извне под составным именем, включающим в качестве префикса имя пакета, например, `P.C`, где `P` — имя пакета, а `C` – имя компонента в этом пакете. Остальные компоненты видимы только внутри данного пакета. В описании компонента пакета видимы все остальные компоненты этого же пакета. Пакет образует собственную область видимости и все компоненты пакета должны иметь несовпадающие имена. В отличие от языков программирования, где компонентами пакета являются только классы, естественными компонентами пакета в ООМ также являются константы, определения типов, алгоритмические функции и процедуры. Кроме того, компонентами пакета могут являться другие пакеты. Общая структура пакета в языке MVL показана на Рис. 35.

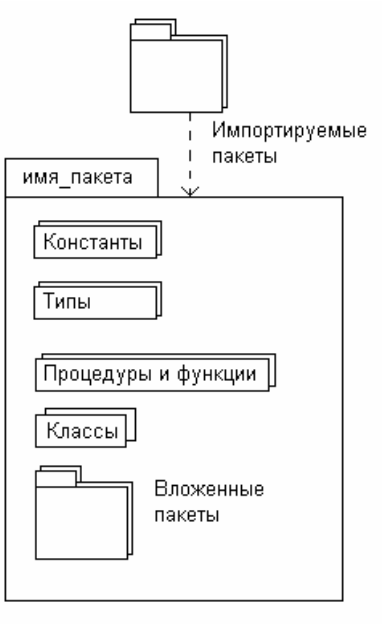


Рис. 35

Для того, чтобы использовать экспортируемые компоненты пакета `A` в пакете `B`, необходимо в писании пакета `B` поместить указание импортирования:

1) «`import A;`» - в этом случае пакет `A` становится видим в пакете `B` и обращения к компонентам пакета `A` должны производиться через указание имени пакета в качестве префикса, например, `A.C1`;
2) «`import A.*;`» - в этом случае все экспортируемые компоненты пакета `A` становятся видимы в пакете `B`;
3) «`import A.C1;`» - в этом случае только конкретный экспортируемый компонент `C1` пакета `A` становятся видимым в пакете `B`.

Конкретный проект является частным случаем пакета, который по умолчанию имеет структуру, показанную на Рис. 36. В новом проекте присутствует только заголовок предопределенного класса `Model`, который является по терминологии UML «синглетным», то есть может иметь только один экземпляр. Этот единственный экземпляр с именем `model` и является выполняемой моделью, с которой проводится вычислительный эксперимент.

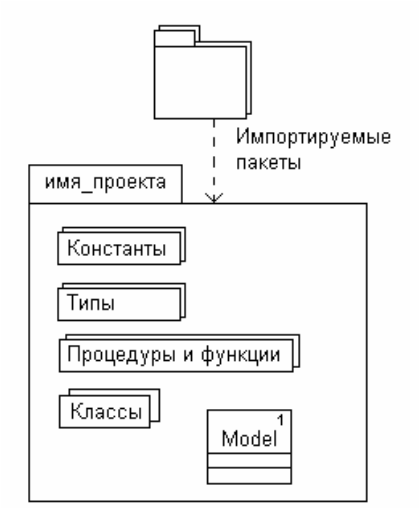


Рис. 36

Если моделируемая система не имеет естественной внутренней структуры, то для проведения вычислительного эксперимента с ней достаточно задать переменные и поведение класса `Model`. Далее отлаженную модель можно сохранить как независимый компонент – класс – в пакете проекта, а класс `Model` использовать для отработки следующего компонента. Для моделей со сложной структурой или поведением можно использовать в качестве составляющих ранее отлаженные компоненты данного пакета или импортировать их из других пакетов.

Например, при создании модели отрывающегося маятника (см. Приложение 1) пользователь может сначала создать и отладить модель колебаний маятника и сохранить ее как класс `Маятник`. Затем пользователь может создать и отладить модель свободного двумерного движения материальной точки, сохранив его затем как класс `ДвижениеXY`. Оба эти класса будут чисто непрерывными элементарными компонентами. Наконец, после этого он может приступить к созданию и отладке модели отрывающегося маятника, задав его поведение в виде карты поведений, включающей состояния `Колебания` и `Свободный_полет`. Этим состояниям можно приписать экземпляры уже готовых классов `Маятник` и `ДвижениеXY`. Компоненты, пригодные для повторного использования, можно переносить из текущего проекта в «тема-

тические» пакеты, играющие роль библиотек классов для определенных прикладных областей (Рис. 35).

Переменные.

Переменные активного динамического объекта являются атрибутами с семантикой значения. Локальные объекты – элементы структурной схемы – являются атрибутами с семантикой указателя. Помимо переменных – атрибутов объекта - в модели могут использоваться также:

- локальные переменные – атрибуты локальных классов;
- рабочие переменные в алгоритмических процедурах и функциях;
- рабочие переменные в описании мгновенных последовательностей действий (действий перехода, входных и выходных действий состояния).

Локальные переменные имеют семантику значения. Рабочие переменные имеют семантику значения, если в качестве типа переменной указан тип данных, и семантику указателя, если в качестве типа переменной указан класс активного или пассивного объекта.

Определение переменной-атрибута в общем случае включает в себя:

- указание видимости;
- стереотип переменной;
- идентификатор переменной;
- указание типа переменной;
- начальное значение переменной.

В определении рабочей переменной отсутствует указание стереотипа.

Указание видимости переменной в MVL ничем не отличается от указания видимости в UML:

- «`public`» или «+» - переменная является внешней;
- «`private`» или «-» - переменная является внутренней и видима только в описании данного класса;

- «protected» или «#» - переменная является внутренней и видима только в описании данного класса и всех его потомков.

Переменные с указанием видимости «public» составляют интерфейс активного динамического объекта, посредством которого он может взаимодействовать с внешним окружением. Остальные переменные по существу являются составляющей описания поведения объекта.

Стереотип переменной отражает семантические правила ее использования:

- «constant» - константа;
- «parameter» - параметр класса. Значение этой переменной может быть изменено только при создании конкретного экземпляра класса;
- «input» - значение переменной не может быть изменено «внутри» объекта;
- «output» - значение переменной может быть изменено только «внутри» объекта;
- «connector» - переменная может участвовать в связях, только переменные с этим стереотипом показываются на изображении объекта в структурной схеме. Этот стереотип предполагает указание видимости «public»;
- «flow» - может использоваться только со стереотипом «connector», указание этого стереотипа предполагает специальный вид уравнений связи.

Стереотип может также указываться в определении типа данных (см. «Явно определяемые типы.»).

Идентификатор переменной представляет собой строку, включающую латинские и русские буквы, цифры и знак подчеркивания и начинающуюся с буквы.

Указание типа переменной представляет собой имя типа (предопределенного, определяемого в пакете или импортируемого) или определение анонимного типа и задает тип значения переменной и возможно стереотип.

Начальное значение переменной представляет собой выражение, в котором могут использоваться и другие переменные (при этом только не должны возникать алгебраические циклы). При создании экземпляра данного класса все переменные приобретают указанные начальные значения или какие-то значения по умолчанию, если в определении класса не указано начальное значение. Значения параметров и начальные значения внешних переменных конкретного экземпляра могут быть явно указаны при вызове конструктора, например

```
P := new Маятник(L:=2, Alpha:=-pi/2, Omega:=1);
```

В этом случае значения, указанные в определении класса, игнорируются.

Типы данных.

Для моделирования непрерывных систем необходим минимальный набор типов данных: скалярный вещественный тип, а также типы «вектор» и «матрица» со своими традиционными операциями, а также целые числа для вычисления индексов векторов и матриц. Для моделирования дискретных и гибридных систем необходимо также иметь более широкий спектр целых типов (байт, короткое целое, длинное целое), перечислимые, булевские, символьные и строковые типы, а также одномерные и двумерные массивы с элементами любого скалярного типа. Кроме того, для описания явной синхронизации параллельных процессов необходимы специфические переменные-сигналы. Для систем со сложной структурой крайне желательно наличие типа «запись». С помощью этого типа очень удобно передавать в компактной форме набор взаимосвязанных данных (возможно различных типов) между структурными компонентами..

Скалярные типы

К скалярным типам относятся вещественный, целые, булевский, перечислимые и символьные типы.

Вещественный тип

Для приближенного представления вещественных чисел в языке MVL используется тип `double`, соответствующий стандарту вычислений с плавающей точкой [70]. С помощью этого типа (внутреннее представление длиной в 8 байт) могут быть с точностью 15-16 значащих десятичных цифр в мантиссе представлены вещественные числа со знаком в диапазоне $5.0 \times 10^{-324} .. 1.7 \times 10^{308}$. Для типа `double` определены следующие операции и отношения: сложение, вычитание, умножение, деление, возведение в целую и вещественную степень, равенство, неравенство, “больше”, “больше или равно”, “меньше”, “меньше или равно”.

Примеры вещественных литералов:

```
-3.5 +5.67 1.5E3 -3.4E12 1.76E-2 4 2e3.
```

Целые типы

В языке MVL поддерживаются три целых типа:

- `byte` (8 бит без знака, диапазон чисел 0.. 255);
- `short` (16 бит со знаком, диапазон чисел -32768 .. 32767):
- `integer` (32 бита со знаком, диапазон чисел -2147483648 .. 2147483647).

Для целых типов определены следующие операции и отношения: сложение, вычитание, умножение, целое деление, сравнение по модулю, возведение в целую степень, побитовое "ИЛИ", побитовое "И", побитовое "НЕ", равенство, неравенство, “больше”, “больше или равно”, “меньше”, “меньше или равно”.

Примеры целых литералов:

```
1 34 -4567
```

Булевский тип

Тип `boolean` имеет множество значений `{false, true}`. Для булевского типа определены следующие операции и отношения: "логическое ИЛИ", "логическое И", "логическое НЕ", равенство, неравенство..

Перечислимые типы

Перечислимые типы определяются путем явного задания (перечисления) конечного множества значений как упорядоченной совокупности не совпадающих по именам литералов вида “(“e_0,e_1, e_2, ..., e_n”)”, где e_i- идентификатор литерала с номером i . Два перечислимых типа являются одинаковыми, если их множества значений совпадают. Для литералов перечислимого типа определены следующие отношения: равенство, неравенство, “больше”, “больше или равно”, “меньше”, “меньше или равно”. Результат этих отношений равен результату соответствующих отношений между номерами значений. Значения различных перечислимых типов несравнимы. Перечислимые литералы задаются своими идентификаторами. Литералы различных перечислимых типов могут иметь совпадающие идентификаторы.

Пример определения перечислимого типа:

```
(Alpha, Beta, Gamma)
```

Символьные типы

К символьным типам относятся собственно символьный тип `char` и строковый тип `string`. Тип `char` включает в себя упорядоченное множество символов. Тип `string` — это строка произвольной длины. Множество символов следует рассматривать как специальный случай перечислимого типа с соответствующими отношениями. Для строк определены следующие операции и отношения: конкатенация, равенство, неравенство. Символьные литералы задаются соответствующим символом, заключенным в кавычки. Примеры символов: `"A"`, `"1"`, `"a"`. Строковые литералы задаются соответствующей строкой, заключенной в кавычки. Примеры строк: `"abcd"`, `"1234"`.

Регулярные типы

К регулярным типам относятся векторы, матрицы и массивы.

Векторы

Определение типа `vector[N]` задает вектор-столбец фиксированной размерности N с элементами типа `double`. Определение типа `vector` задает вектор-столбец переменной размерности с элементами типа `double`. Элементы вектора всегда нумеруются с 1. Контроль правильности использования вектора с переменной размерностью возможен только во время исполнения. Текущую размерность вектора всегда можно определить с помощью функции `size()`. Для векторов определены следующие операции и отношения: умножение на скаляр, сложение, вычитание, транспонирование, равенство, неравенство.

Примеры векторных литералов:

`[1;2;3;4]`, `[0; 0; 2.3; 5.67; 1E2]`.

`[for i in 1..10 | i**2 ]` — вектор размерности 10, содержащий значения 1, 2, 9, …, 100 (см. итеративный матричный литерал).

Матрицы

Определение типа `matrix[N,M]` задает прямоугольную матрицу фиксированной размерности с N строками и M столбцами с элементами типа `double`. Определение типа `matrix` задает прямоугольную матрицу переменной размерности. Элементы матрицы всегда нумеруются с 1 по обоим измерениям. Вектор всегда можно рассматривать как матрицу [N,1]. Контроль правильности использования матрицы с переменной размерностью возможен только во время исполнения. Текущую размерность матрицы всегда можно определить с помощью функции `size()`. Для матриц определены следующие операции и отношения: умножение матрицы на скаляр, умножение матриц, сложение, вычитание, транспонирование, равенство, неравенство.

Примеры матричных литералов:

`[1, 2, 3, 4; 1, 4, 5, 6]` — матрица 2×4;

`[0,3.4,5;7,8,0.67;0.8,6,2.3]` — матрица 3×3.

Итеративные матричные литералы можно использовать в качестве начального значения переменных, а также в качестве правой части формулы или оператора присваивания.

Массивы.

В MVL поддерживаются одномерные и двумерные массивы с элементами какого-либо скалярного типа. Границы индексации элементов указываются явно в виде диапазона. Примеры определений массивов:

```
array [1..3] of boolean;
array [0..4, 1..2] of integer;
```

Примеры литералов - массивов:

```
(true,false,true)
(0,1,2,3,4,
 0,0,0,1,1)
```

Комбинированный тип (запись).

Переменная комбинированного типа есть последовательность поименованных компонент. Компоненты записи могут принадлежать к различным типам (рекурсии в определении записи не допускаются). Примеры определений записей:

```
record
  A: integer;
  B: boolean;
  C: matrix[2,3];
end record;
```

Примеры комбинированных литералов:

```
(A=>2, B=>true, C=>[1,2,3; 0,0,1])
```

Явно определяемые типы.

Декларация типа позволяет связать идентификатор типа с некоторым определением типа и в дальнейшем использовать этот идентификатор для задания

типа констант, переменных и формальных параметров. Примеры деклараций типа:

```
type T1 is integer;
type T2 is matrix[2,2];
type T3 is (A,B,C);
type T1_1 is T1;
type T4 is record
            X: T2;
            Y: double;
            Z: array [0..2] of boolean;
            W: T3;
           end record;
```

Литералы определяемого типа соответствуют его определению:

```
V1: T1 := 2;
V2: T2 := [1,2;2,4];
V3: T3 := B;
V4: T4 := (X=>[0,1;1,0], Y=>4.67, Z=>(false,false,true), W=>A);
```

В определении типа может также быть указан стереотип переменных, имеющих значения данного типа, например:

```
type Voltage is double;
type Current is double;
connector type Pin is
     record
       V: Voltage;
       flow I: Current;
     end record;
```

Переменная, декларированная как «`N: Pin;`» автоматически приобретает стереотип «`connector`».

Сигналы.

Как отмечалось выше, переменные типа `signal` – это сообщения (возможно с параметрами), передаваемые между параллельно выполняемыми процессами с целью их явной синхронизации. Формальные параметры сигнала должны декларироваться в определении переменной или типа, например:

```
Throw: signal (V: double; Teta: double);
```

```
type Throw is signal V: double; Teta: double);
SignalA: signal;
```

С переменной-сигналом можно выполнить только одно действие: послать сигнал с помощью оператора `send`, указав фактическое значение параметров, например:

```
send Throw (V:=100; Teta:=rad(45));
```

Фактические значения параметров сигнала доступны только для чтения в мгновенных действиях перехода, который принял данный сигнал, через составные имена с именем сигнала в качестве префикса, например:

```
Vx := Throw.V*cos(Throw.Teta);
```

При использовании регулярных структур компонентов возможно появление массивов сигналов.

Автоматическое приведение типов

Автоматическое приведение типов производится при использовании в операциях или отношениях операндов различных типов в следующих случаях:

- при использовании различных целых типов операнд меньшей разрядности приводится к операнду большей разрядности;
- при использовании целого и вещественного типов целое значение приводится к вещественному типу;
- при использовании символьного и строкового типов символьное значение приводится к соответствующей строке длиной 1;
- типы `vector[1]` и `matrix[1,1]` могут трактоваться как `double` и наоборот.

Правила приведения типов действуют также при соединении внешних переменных компонент направленными связями (тип источника приводится к типу приемника).

Система уравнений.

Система уравнений соответствует «внешней» математической модели непрерывной системы, введенной в Главе 2, то есть в общем случае это сис-

тема дифференциально-алгебраических уравнений, в которой используются производные порядка выше первого и не разрешенная относительно производных. Кроме того, в языке MVL система уравнений может включать в себя формулы невещественного типа, а также условные уравнения. В общем случае эти конструкции требуют предварительного преобразования к эквивалентному гибридному автомату (см. раздел 0). В частном случае, когда используются только направленные компоненты и набор искомых переменных не изменяется при переключении ветвей условного выражения, условные уравнения могут непосредственно поддерживаться исполняющей системой пакета моделирования (см. Главу 4).

Карта поведений.

«Дискретная составляющая» карты поведений, определяющая правила «склейки» локальных поведений, почти полностью соответствует «машине состояний», определенной в языке UML:

- имеются начальное и конечное состояния;
- допускаются гиперсостояния;
- описание состояния включает в себя входные и выходные мгновенные действия, а также выполняемую деятельность и внутренние переходы;
- описание перехода включает в себя исходное и конечное состояния, запускающее событие, охраняющее условие и последовательность мгновенных действий;
- для удобного задания логики дискретных действий используется условный переход.

Все различия обусловлены тем, что под «деятельностью» («`activity`») в карте поведений понимается не последовательность (разовая или циклическая) дискретных действий, а некоторый активный динамический объект X, создаваемый при входе в данное состояние (после выполнения входных действий) и уничтожаемый при выходе из данного состояния (после выполнения выходных действий). Если данное состояние является текущим на некотором

интервале гибридного времени, то в течение этого интервала совокупное поведение объекта, которому принадлежит гибридная карта состояний, является результатом объединения его собственного поведения с поведением объекта X.

Объект-деятельность X может быть:

- экземпляром локального класса;
- экземпляром класса, определенного в данном пакете;
- экземпляром класса, импортированного из другого пакета;
- композицией нескольких экземпляров (см. пример 4 в Приложении 1).

Такое определение деятельности накладывает ряд ограничений даже в случае чисто дискретной карты поведений. Различия прежде всего связаны с трактовкой подсостояний. Многоуровневая внешне, карта состояний UML по существу является плоской одноуровневой, так как разрешается задавать «прямые» переходы извне непосредственно на вложенное подстостояние и наоборот из подсостояния на состояние верхнего уровня иерархии. Иерархическая карта поведений получается простым использованием дискретных или гибридных компонентов в качестве деятельностей. В этом случае мы имеем дело с действительно иерархической вложенностью. Очевидно, что никакие «прямые» переходы здесь невозможны, поскольку поведение инкапсулировано внутри объекта X. При создании экземпляра локальной карты поведений ее текущим состоянием всегда является начальное. Соответственно, невозможны и переходы в так называемое «историческое» состояние, поскольку вложенная карта поведений просто уничтожается при выходе из охватывающего состояния.

Еще две конструкции классической карты состояний UML - параллельные подстостояния и соединение/разъединение переходов – не могут использоваться в карте поведений так как противоречат синхронному принципу объединения гибридных автоматов.

Таким образом, язык MVL позволяет строить сложные поведения путем последовательной (а не параллельной, как в случае структурной схемы)

композиции отдельных компонентов – объектов-деятельностей -, взаимодействующих через начальные условия. Так, например, сложное поведение отрывающегося маятника может строиться как последовательная композиция двух простых непрерывных объектов – модели колебаний маятника и модели свободного двумерного движения материальной точки (см. пример 2 в Приложении 2). Эти модели имеют различные наборы переменных и системы уравнений. Для того, чтобы согласовать эти две модели, необходимо в момент отрыва по текущим значениям угла и угловой скорости маятника вычислить линейные координаты и скорости и использовать их в качестве начальных значений для модели свободного движения. Сам момент отрыва определяется логическим предикатом, зависящим от переменных модели маятника, например, от угла. Кроме того, для анимации модели отрывающегося маятника постоянно необходимы линейные координаты. Таким образом, при создании сложного поведения путем последовательной композиции объектов возникает необходимость использования видимых переменных объектов-деятельностей в условиях переходов, мгновенных действиях, параметрах конструктора объекта-деятельности следующего состояния и в уравнениях, относящихся к охватывающему объекту.

Если объект-деятельность является экземпляром локального класса, то в описании его поведения видимы все переменные-атрибуты охватывающего класса, которые могут и играть роль внешних переменных объекта-деятельности. Если же деятельность является экземпляром внешнего класса, то для обеспечения непрерывной связи с переменными-атрибутами, рабочими переменными карты состояния или данного состояния необходимо добавить в определение деятельности дополнительные уравнения связи (см. пример 2 в Приложении 2).

Компоненты объекта-деятельности невидимы вовне данного состояния. Поэтому условие срабатывания исходящего из данного состояния перехода может быть выражено либо через видимые в переходе переменные, либо через сигнал, которые вырабатывается внутренним переходом в текущем со-

стоянии. Объект-деятельность существует от момента окончания входных действий в текущем состоянии до момента окончания выходных действий в этом состоянии. Поэтому передача информации следующему состоянию возможна через видимые в обоих состояниях переменные, которым присваиваются значения:

- в выходных действиях данного состояния, где видимы компоненты объекта-деятельности;
- в действиях перехода, куда необходимая информация должна поступать в качестве параметров сигнала.

Формализм обобщенного гибридного автомата позволяет использовать «исчисление» поведений. Например, в примере 4 Приложения 1 деятельность определяется как

```
БазоваяСистемаУравнений + {Ia = 0}
```

Первый член этого выражения является анонимным экземпляром локального непрерывного класса. Второй член является анонимным экземпляром анонимного локального непрерывного класса, задаваемого неявно. Композиция этих объектов дает в результате анонимный непрерывный объект, который и является деятельностью.

Структурная схема.

Структурная схема включает в себя описания экземпляров локальных активных объектов и их связей, то есть задает совокупность параллельно функционирующих компонентов и их явные взаимодействия. Взаимодействие между локальными объектами также может осуществляться и неявным образом - через собственное поведение (уравнения и гибридную карту состояний) объекта-контейнера.

Объекты.

Локальный объект может являться экземпляром класса, определенного в данном проекте или импортированного из других пакетов. Локальный объект не может являться экземпляром локального класса, так как в этом случае

не гарантируется синхронное объединение гибридных автоматов. Визуальным образом локального объекта на структурной схеме является прямоугольник, на границах которого условно изображаются переменные со стереотипом «connector». Локальные объекты изображаются внутри большого прямоугольника, соответствующему гипотетическому экземпляру класса-контейнера (Рис. 37).

Внешнюю переменную со стереотипом «connector» будем называть «разъемом». Соответственно переменную со стереотипом «connector input» будем называть входом, а переменную со стереотипом «connector output» будем называть выходом. Эти виды разъемов являются направленными и изображаются на структурной схеме стрелками соответствующего направления. Ненаправленные разъемы будем также называть «контактами». Контакты изображаются на структурной схеме квадратами (Рис. 37). Контакт со стереотипом «flow» будем называть «потоком».

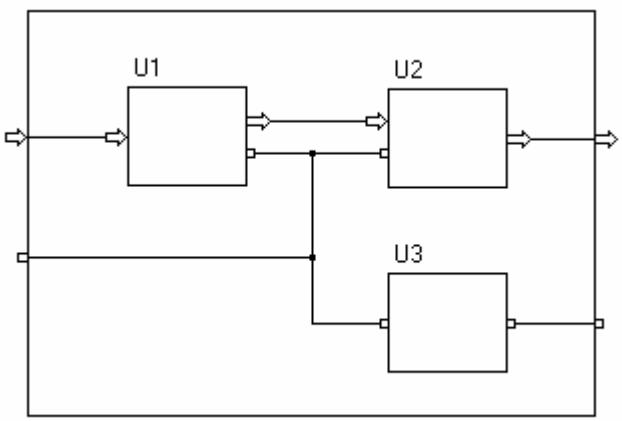


Рис. 37

Связи.

Связь или соединение является указанием стандартного взаимодействия, имеющего свой графический образ на структурной схеме. Связь изображается линией, соединяющей два «разъема» (Рис. 37). Соответственно типу соединяемых разъемов связь является направленной или ненаправленной.

Совокупности связей соответствует система уравнений и формул, неявно добавляемая к системе уравнений класса-контейнера.

Направленным связям, соединяющим направленные разъемы A, B и C (Рис. 38),

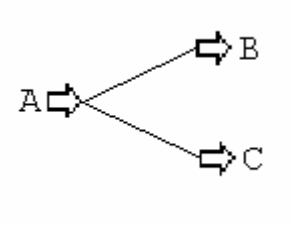


Рис. 38

соответствуют *формулы* $\begin{cases} B = A \\ C = A \end{cases}$. Разъем A может являться выходом локального объекта или входом объекта-контейнера, соответственно разъемы B,C могут являться входами локальных объектов или выходами объекта-контейнера. Переменные A, B и C могут иметь любой тип.

Ненаправленным связям, соединяющим контакты A, B и C (Рис. 39),

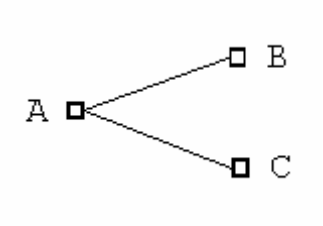


Рис. 39

соответствуют *алгебраические уравнения*:

- если эти контакты не являются потоками, то $\begin{cases} A - B = 0 \\ A - C = 0 \end{cases}$;
- если эти контакты являются потоками, то $\{A + B + C = 0$

Переменные A, B и C могут иметь тип `double`, `vector` или `matrix`., а также `record` с полями перечисленных типов. В последнем случае уравнения связей формируются покомпонентно.

Регулярная структура.

Иногда возникает необходимость в использовании регулярных структур локальных объектов. Например, мы можем определить зависимость дальности падения тела, брошенного под углом к горизонту, от угла бросания, бросив 17 тел одновременно под углами 5, 10, 15, …, 85 градусов и замерив координаты точек их падения. Для проведения такого эксперимента нам потребуется массив экземпляров класса ДвижениеXY

```
Тела: array [1..17] of ДвижениеXY := {for i in 1..17 | new
                    ДвижениеXY(Teta0:=rad(5*i)}
```

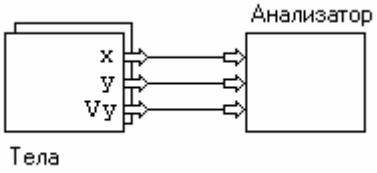


Рис. 40

На структурной схеме этот массив будет изображаться как мультиобъект (Рис. 40). В описании поведения класса-контейнера доступен как весь массив, так и его элементы. В связях может участвовать только мультиобъект целиком (Рис. 40). Следует отметить, что переменные мультиобъекта будут также иметь регулярный тип, например, переменная `Тела.Vy` будет иметь тип `array[1..17] of double` или `vector[17]`.

Переменная структура.

Обычно предполагается, что локальные объекты являются статическими, то есть они автоматически создаются при создании экземпляра класса-контейнера и автоматически уничтожаются при уничтожении этого экземпляра. Так как самым верхним в иерархии объектов является экземпляр класса `Model`, существующий в течение всего прогона модели, то и статические локальные объекты существуют в течение всего прогона модели. В этом случае можно не различать объект и указатель (ссылку) на объект. Все мультиобъекты, показанные на структурной схеме сплошными линиями, являются статическими.

Однако, в некоторых случаях оказываются необходимы переменные – указатели на объекты и мультиобъекты с переменным составом.. Модифицируем предыдущий пример. Будем бросать 17 тел не одновременно, а друг за другом с интервалом в 1 секунду, увеличивая каждый раз угол бросания на 5 градусов. Ясно, что число одновременно летящих тел будет переменным. Используем локальный мультиобъект с переменным составом - список `ЛетящиеТела` (Рис. 41). Указатели на объекты и мультиобъекты с переменным составом изображаются на структурной схеме пунктирными линиями.

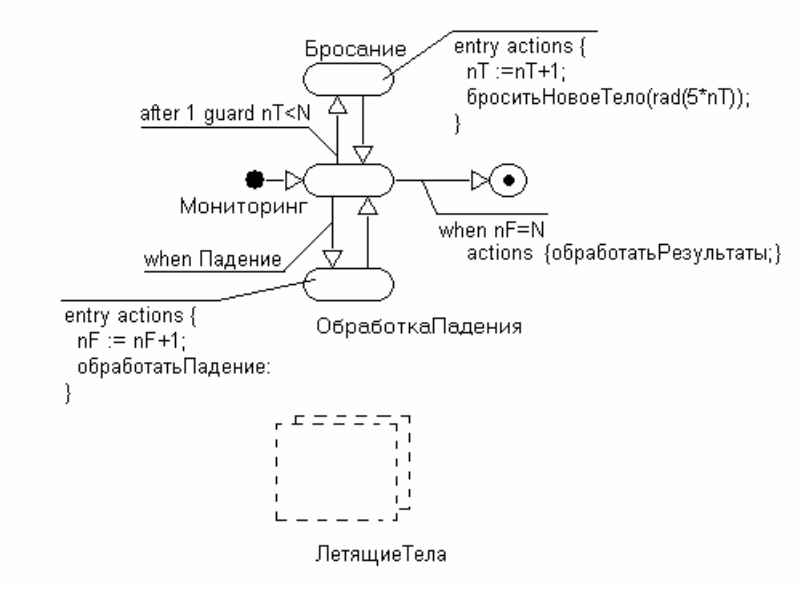


Рис. 41

В карте состояний объекта-контейнера, показанной на Рис. 41, используются алгоритмические процедуры и функции:

```
procedure броситьНовоеТело (Teta: double) is
  НовоеТело: ДвижениеXY;
begin
  НовоеТело := new ДвижениеXY (V0:=V; Teta0:=Teta);
  ЛетящиеТела.add(НовоеТело);
  Тела.add(НовоеТело);
end броситьНовоеТело;

function Падение return boolean is
  X: ДвижениеXY;
  res: boolean := false;
```

```
begin
  for i in 0..ЛетящиеТела.size-1 loop
    X := ЛетящиеТела.get(i);
    res := (X.y<=0) and (X.Vy<0);
    exit when res;
  end loop;
  return res;
end Падение;

procedure обработатьПадение is
  X: ДвижениеXY;
begin
  for i in 0..ЛетящиеТела.size-1 loop
    X := ЛетящиеТела.get(i);
    exit when (X.y<=0) and (X.Vy<0);
  end loop;
  ЛетящиеТела.remove(X);
end обработатьПадение;
```

Правила видимости.

Правила видимости определяют область действия описаний и устанавливают, какие идентификаторы видимы в различных местах проекта. Описание сопоставляет идентификатор некоторому декларативному элементу проекта (классу, переменной, функции и т.п.). Часть проекта, в которой сказывается влияние какого-либо описания, называется областью действия этого описания. Один и тот же идентификатор может быть введен с помощью разных описаний и, следовательно, сопоставлен различным элементам проекта. Из-за иерархической блочной структуры проекта области действия этих описаний могут перекрываться. Описание некоторого элемента проекта *E* с идентификатором *I видимо* в данном месте проекта, если идентификатор *I* может обозначать здесь элемент *E*. В языке MVL в любом месте проекта идентификатор может обозначать не более одного элемента. Описание элемента E1 видимо в своем блоке и во всех блоках, вложенных в данный. Если во вложенном блоке имеется описание другого элемента E2 с тем же идентификатором

I, что и у E1 , то элемент E1 *скрыт* в области действия элемента E2, т.е. идентификатор I обозначает здесь элемент E2. Например, если в проекте объявлена константа X1 типа string, а в описании некоторго класса обявлен вход X1 типа double, то везде в пределах описания данного класса под X1 будет пониматься именно вход типа double.

Областями видимости являются (Рис. 42):

- проект в целом;
- описание класса;
- алгоритмическая процедура или функция;
- локальный класс;
- мгновенные действия (входные/выходные действия в состоянии, действия в переходе).

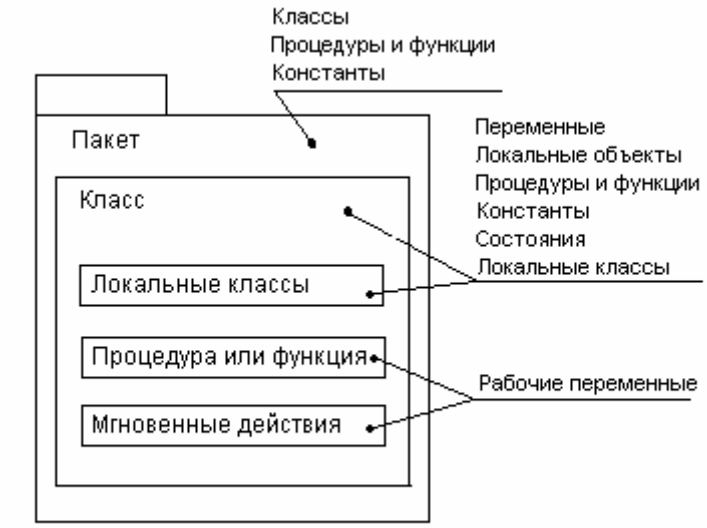


Рис. 42

Кроме того:

- в пределах оператора цикла **for** видим параметр цикла;
- в пределах регулярного выражения видимы параметры цикла;
- в пределах алгоритмической процедуры или функции видимы формальные параметры;
- в пределах мгновенных действий перехода с сигналом в качестве запускающего события видимы формальные параметры этого сигнала.

В пределах одной области видимости порядок деклараций элементов безразличен, то есть в текстовом представлении в определении элемента могут использоваться другие, определенные ниже (например, в выражении для начального значения переменной могут использоваться другие переменные и функции, определенные ниже).

Наследование классов.

С помощью наследования можно обеспечивать повторное использование ранее разработанных и отлаженных моделей. Если класс `C2` объявляется прямым потомком класса `C1` (Рис. 43), то класс `C2` наследует все элементы класса `C1`: переменные, процедуры и функции, локальные классы, карту поведений, систему уравнений и структурную схему. Применительно к отношению `C1`←→`C2` класс `C1` будем далее называть базовым классом или суперклассом, а класс `C2` – производным классом или подклассом. Отношение наследования транзитивно: если класс `C1` сам является производным от некоторого базового класса (на Рис. 43 это класс `C0`), то и класс `C2` является производным от класса `C0` и наследует все его элементы.

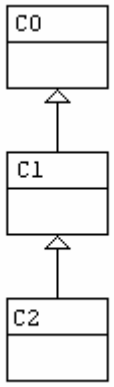


Рис. 43

Кроме того, все изменения, вносимые в класс `C1` и в любой из его предков, будут автоматически отражаться на классе `C2`. В языке MVL допустимо только одиночное наследование: любой производный класс может иметь только один базовый.

Целью наследования является расширение и/или модификация описания базового класса. Это можно осуществить с помощью добавления новых элементов описания и переопределения унаследованных элементов описания. Никакие унаследованные элементы не могут быть удалены. Все активные динамические объекты являются потомками предопределенного класса `ActiveDynamicObject`.

Добавление новых элементов описания.

В языке MVL, как и в Modelica, новые элементы описания не могут иметь имена, совпадающие с элементами описания базового класса. Это противоречит традиционному для объектно-ориентированных языков программирования правилу, согласно которому новая переменная с тем же именем, что и унаследованная, скрывает последнюю в описании производного класса. Однако, в описании активного динамического объекта, в отличие от пассивного объекта, оба элемента (например, переменные) могут использоваться одновременно – в системе уравнений, к которой добавлены новые уравнения или в карте поведений, к которой добавлены новые состояния и переходы. Такая ситуация совершенно недопустима, так как может привести к неверному пониманию пользователя и провоцировать ошибки.

В производном классе могут быть определены новые переменные, процедуры и функции, а также новые локальные классы. В систему уравнений могут быть добавлены новые уравнения. В карту поведений могут быть добавлены новые состояния и переходы. В структурную схему могут быть добавлены новые локальные объекты и новые связи.

Переопределение унаследованных элементов.

В описании производного класса может быть переопределено определение унаследованного элемента. Переопределение является основным механизмом модификации базового класса. Традиционно переопределяемый элемент определяется по совпадению имени. Однако, в гибридных моделях имеются конструкции, не имеющие имен: переходы и уравнения. В пакете

MVS для хранения визуального описания проекта используется объектно-ориентированная база данных (см. Главу 4), в которой каждый хранимый объект по определению имеет собственный уникальный идентификатор [4]. Этот идентификатор, соответствующий визуальному представлению языковых конструкций («этот переход», «вот это уравнение»), и используется инкрементным транслятором MVS для определения переопределяемого элемента. При создании текстового представления проекта для всех неименованных конструкций автоматически генерируются служебные имена, которые пользователь может использовать при редактировании проекта в текстовом представлении. Пакетный компилятор из текстового представления в визуальное также использует имена.

Переопределение переменных.

При переопределении переменной можно изменить только ее стереотип. Таким способом можно, например, параметр сделать входной переменной или «разъемом». Возможность изменения стереотипа позволяет гибко адаптировать существующие классы к особенностям конкретных моделей.

Переопределение процедур и функций.

При переопределении процедуры или функции должна быть полностью сохранена унаследованная сигнатура (список параметров и тип результата). Для дополнительных параметров должны быть обязательно указаны значения, принимаемые по умолчанию. Тело процедуры или функции изменяется полностью (хотя редактор в качестве начального приближения предлагает унаследованный текст, всякая связь с телом соответствующей функции в базовом классе утрачивается).

Переопределение локального класса.

При переопределении локального класса в него могут быть добавлены новые элементы и переопределены унаследованные.

Переопределение элементов карты поведений.

При переопределении карты состояний можно:

- заменить входные/выходные действия в состоянии;
- заменить деятельность в состоянии;
- заменить условие срабатывания и/или охраняющее условие перехода;
- заменить последовательность действий в переходе;
- изменить графическое изображение состояния или перехода.

Переопределение отдельных уравнений.

Конкретное уравнение в унаследованной системе уравнений можно заменить на другое уравнение (см. пример 3 в Приложении 2). Для пользователя при работе в интегрированной оболочке соответствие уравнений будет чисто визуальным: «заменить вот это уравнение».

Переопределение элементов структурной схемы.

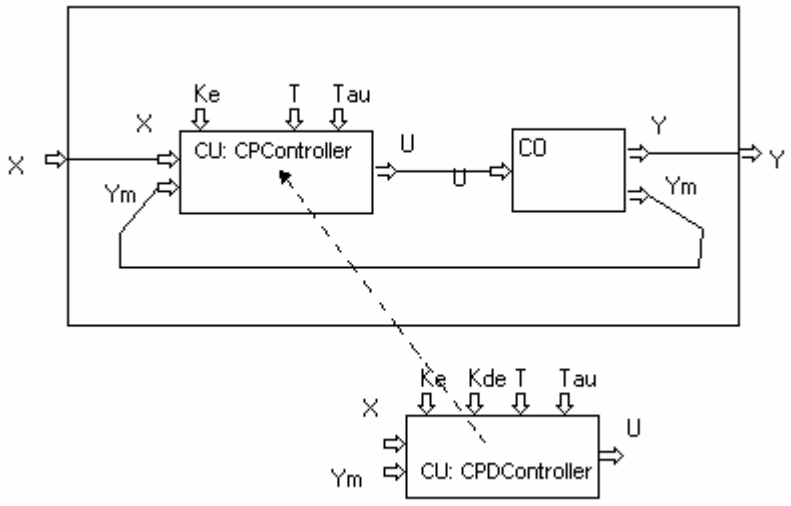


Рис. 44

При переопределении структурной схемы можно заменить локальный объект на любой другой, в котором существуют внешние переменные, соответствующие по идентификатору и типу значения внешним переменным заменяемого объекта. Например, на Рис. 44 показана модификация схемы сис-

темы автоматического регулирования путем замены P-регулятора на PD-регулятор.

Преобразование системы уравнений или карты поведений в локальный класс.

Необходимость в таком преобразовании возникает при модификации карты поведений в производном классе, при которой система уравнений или карта поведений базового класса используются как деятельность в определенных состояниях. Например, пусть мы хотим на основе класса `ДвижениеXY`, описывающего неограниченное двумерное движение в поле тяготения, создать модель снаряда, то есть тела, которое после контакта с горизонтальной плоскостью остается в точке падения. Система уравнений класса `ДвижениеXY` должна быть деятельностью в состоянии `Полет` гибридной карты состояний (Рис. 45). В противном случае система уравнений будет продолжать решаться и после перехода в конечное состояние, что неверно.

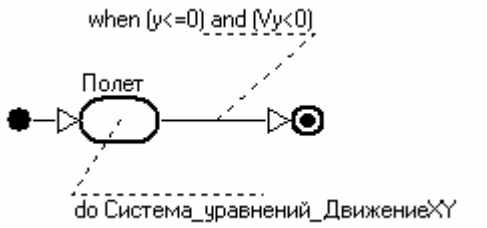


Рис. 45

Поэтому вся унаследованная система уравнений должна быть преобразована в локальный класс `Система_уравнений_ДвижениеXY`, экземпляр которого используется в качестве деятельности. Система уравнений производного класса становится пустой. Однако, унаследованная система уравнений при этом не удаляется, а только преобразуется.

Полиморфизм.

В языке MVL используется как «традиционный» полиморфизм объектов, так и полиморфизм «по интерфейсу», характерный для языка Modelica . «Традиционный» полиморфизм заключается в том, что вместо декларированного объекта определенного класса C фактически могут использоваться экземпляры любых производных от C классов. Полиморфизм «по интерфейсу» означает, что можно заменить локальный объект на любой другой, в котором существуют внешние переменные, соответствующие по идентификатору и типу значения внешним переменным заменяемого объекта (см. Рис. 44).

Язык управления экспериментом.

Часто вычислительный эксперимент не сводится к однократному получению фазовой траектории моделируемой системы, а требует многократных «прогонов» модели при различных начальных условиях согласно определенной – часто довольно сложной - логике. Для задания этой логики необходимо каким-то образом задать алгоритм эксперимента.

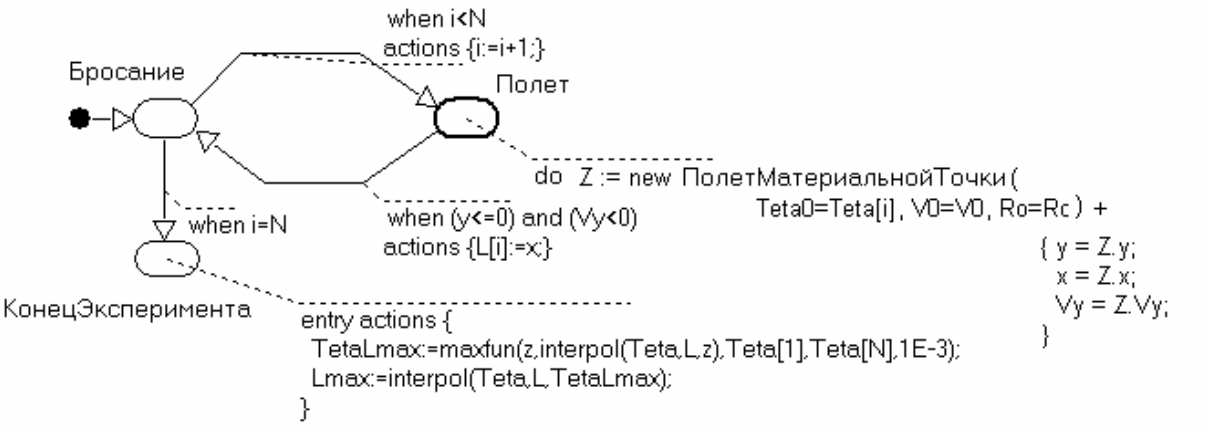
Пусть, например, мы имеем модель тела, брошенного под углом к горизонту, и хотим получить зависимость дальности падения тела от угла бросания. Далее мы хотим исследовать, как зависит угол максимальной дальности от плотности воздуха и от начальной скорости тела. Для получения требуемых результатов экспериментатору необходимо:

- N раз провести моделирование полета тела до момента падения для N различных значений угла бросания;
- определить угол максимальной дальности и запомнить его;
- M раз повторить предыдущие пункты для различных значений плотности воздуха и т.д.

Для задания алгоритма проведения вычислительного эксперимента часто используются процедурные языки (в работе [90] описывается использование для управления экспериментом входного языка пакета Mathematica, имеются попытки использования для этих целей интерпретируемого языка програм-

мирования Python [6]). Однако, здесь нельзя использовать только стандартные алгоритмические конструкции, поскольку необходимы средства синхронизации с процессами, протекающими в модели (в данном примере необходимо определить момент падения тела, то есть синхронизироваться с дискретным событием). Между тем, удобный визуальный язык для описания взаимодействия с параллельными процессами, протекающими в непрерывном времени, хорошо известен. Это язык карт состояний. Введенное в данной работе его расширение – обобщенный гибридный автомат – может быть непосредственно использовано для управления вычислительным экспериментом.

Идея состоит в том, что модель исследуемой системы становится составляющей локального поведения в одном или нескольких состояниях гибридного автомата, задающего алгоритм проведения вычислительного эксперимента. Сам этот автомат является элементом поведения предопределенного объекта «`Model`». Рассмотрим предлагаемую технологию на приведенном выше примере – исследование поведения тела, брошенного под углом к горизонту.




```
parameter Teta: vector;
parameter V0: double;
parameter Ro: double;
N: integer := size(Teta);
L: vector := zeros(N);
i: integer := 0;
TetaLmax,Lmax: double;
x,y,Vy: double;
```

Рис. 46

Сначала мы должны создать и отладить саму модель материальной точки с массой m, брошенной с начальной скоростью V_0 под углом θ_0 к горизонту в воздушной среде с постоянной плотностью ρ и плоскопараллельном гравитационном поле с ускорением силы тяжести g. Очевидно, что эту модель естественно строить как модель изолированной системы. Пусть это будет простейшая модель неограниченного движения, включающая только уравнения движения материальной точки. Отлаженную модель сохраним как класс `ПолетМатериальнойТочки` с параметрами θ_0, V_0 и ρ.

Алгоритм построения зависимости дальности полета от угла бросания и определения угла максимальной дальности реализуется несложной картой поведений, показанной на Рис. 46. Экземпляр `Z` класса `ПолетМатериальнойТочки` создается при входе в состояние `Полет` и уничтожается при выходе. При этом параметр θ_0 этого экземпляра принимает значение текущего угла бросания в эксперименте `Teta[i]`. Кроме того, к уравнениям движения материальной точки добавляются три уравнения связи, отражающие «погружение» компоненты `Z` в контекст компоненты «Виртуальный стенд». Алгоритм проведения эксперимента предусматривает столько «прогонов» модели `ПолетМатериальнойТочки`, какова размерность массива пробных углов `Teta`. После падения очередной материальной точки горизонтальная координата падения запоминается в `L[i]`. По окончании эксперимента с помощью интерполяции определяются угол `TetaLmax`, при котором достигается максимальная дальность, и само значение максимальной дальности `Lmax`.

Таким образом, благодаря использованию обобщенного гибридного автомата и «исчисления поведений» удается представить алгоритм управления вычислительным экспериментом в предельно наглядной визуальной форме.
Сохраним автомат управления экспериментом как класс `ОпределениеОптимальногоУгла`. Теперь, используем экземпляр этого класса как составляющую локального поведения в состоянии гибридного автомата следующего уровня иерархии, предназначенного для управления вычислительным экс-

периментом, целью которого является получение зависимости $\theta_{\max}(V_0, \rho)$ (Рис. 47).

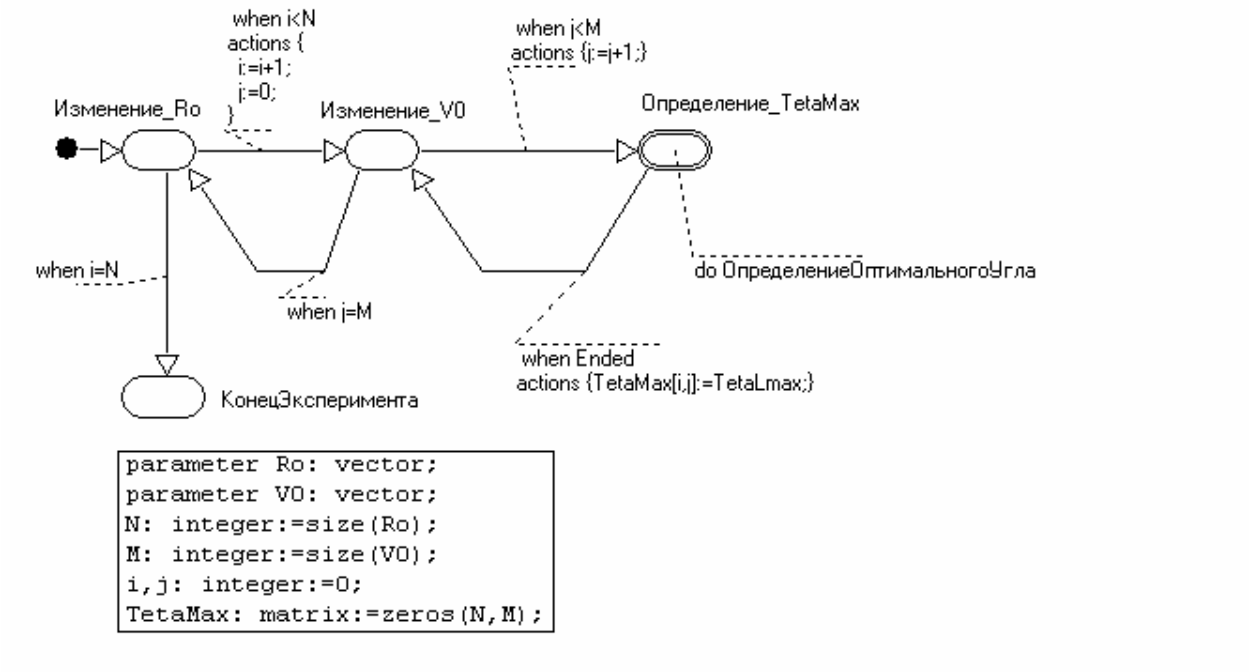


Рис. 47

Функциональный стиль моделирования.

Рассмотрим модель изолированной системы, поведение которой задается уравнениями

$$\begin{cases} \dfrac{dy}{dt} = Saturation(x, -1, 1) \\ x = A \cdot \sin(\omega \cdot t) \end{cases}$$

где функция $Saturation(x, LowerLimit, UpperLimit)$ задает зависимость, показанную на Рис. 48.

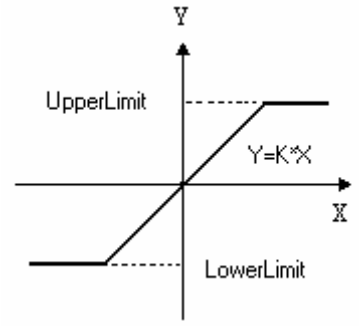


Рис. 48

Конечно, можно рассматривать эту функцию как чисто алгоритмическую и задать ее, например, в виде

```
double Saturation(double x, double LowerLimit, double UpperLimit) {
  if (x>UpperLimit) return UpperLimit;
  else if (x<LowerLimit) return LowerLimit;
  else return x;
}
```

Многие модельеры так и поступают, однако, в этом случае мы имеем систему уравнений с разрывной второй производной от y. Конечно, большинство численных методов благополучно справляются с этой проблемой. Однако, в случае, когда данная система объединяется с другими, это далеко не очевидно. Другие функции, например, релейная, показанная на Рис. 49, дают разрывы в первой производной

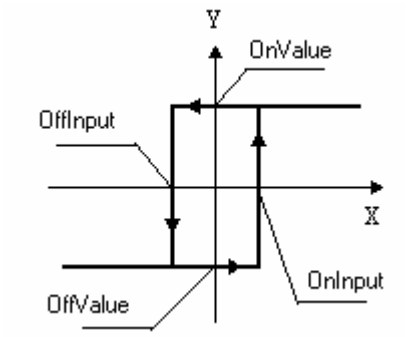


Рис. 49

Другим решением является явная запись тела функции в виде условного уравнения

$$\begin{cases} \dfrac{dy}{dt} = if\ x > UpperLimit\ then\ UpperLimit\ else\ if\ x < LowerLimit\ then\ LowerLimit\ else\ x \\ x = A \cdot \sin(\omega \cdot t) \end{cases}$$

Такая запись даст корректное численное решение: при переключении одной ветви условного выражения на другую генерируется дискретное событие, интегрирование прекращается, находятся новые начальные условия и интегрирование стартует заново. Однако, даже для этой простой функции явное выписывание ее алгоритма при каждом использовании не очень удобно, а для

более сложных функций просто затруднительно. Например, алгоритм релейной функции (Рис. 49) нельзя записать в виде простого условного выражения без памяти.

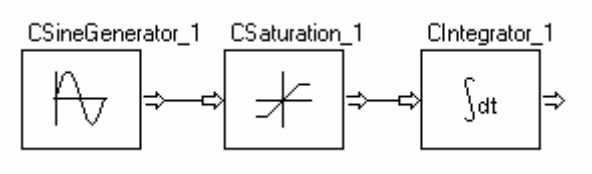


Рис. 50

Наконец, третьим решением является представление системы уравнений в виде структурной схемы (Рис. 50), в которой стандартный блок Saturation определен корректным образом (с помощью условных уравнений или гибридного автомата). Однако, для построения этой схемы нам пришлось разбить нашу простую систему уравнений на две части, создать два локальных блока (в данном случае оказалось возможным использовать стандартные блоки) и соединить их через блок звена с насыщением. Для сложных систем уравнений, в которых используется много разрывных функций, может потребоваться довольно запутанная схема, при составлении которой будут сделаны новые ошибки. Кроме того, эта схема является искусственной, не отражающей естественной структуры моделируемой системы.

Поэтому хотелось бы иметь возможность максимально использовать «функциональный стиль» в определении поведения изолированной системы, обеспечивая в то же время корректное численное решение. Следует отметить, что «опасными» с точки зрения численного решения являются также и некоторые общематематические функции, такие как `abs`, `sign`. Решением этой проблемы является использование компонентов-функций. Определим для всех стандартных блоков с разрывами (звено с насыщением, релейное звено, зона нечувствительности, генератор «пилы», генератор импульсов и т.д.) соответствующие стандартные функции. Создадим также для всех «опасных» математических функций соответствующие стандартные блоки. Например, блок `Abs` с входом `x` и выходом `y` будет задаваться уравнением

$$y = if\ \ x \geq 0\ then\ x\ else - x$$

Задание функции уравнением принципиально отличается от задания этой функции оператором присваивания (пусть и текстуально совпадающим): при решении условного уравнения при переключении ветвей условного выражения будут генерироваться соответствующие дискретные события. Теперь транслятор, обрабатывая уравнения вида

$$\begin{cases} \dfrac{dy}{dt} = Saturation(x = x, LowerLimit = -1, UpperLimit = 1) \\ x = A \cdot \sin(\omega \cdot t) \end{cases}$$

должен автоматически преобразовать эту непрерывную систему к компоненту-контейнеру с локальным компонентом

```
S1: Saturation := new Saturation(LowerLimit=-1,UpperLimit=1);
```

и собственным поведением

$$\begin{cases} \dfrac{dy}{dt} = S1.y \\ S1.x = x \\ x = A \cdot \sin(\omega \cdot t) \\ \end{cases}$$

показанным на Рис. 51.

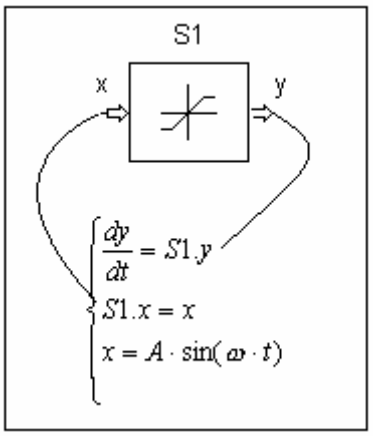


Рис. 51

В случае использования в уравнениях нескольких компонент-функций, создается соответствующее число локальных экземпляров этих компонент. Для случаев, когда компонента-функция имеет единственный выход, результат функции естественным образом ассоциируется с его значением. Если компо-

нента функция имеет несколько выходов, то предполагается, что результат функции имеет тип `record` и в уравнениях необходимо указывать нужную составляющую результата.

Часто бывает удобно включать в систему уравнений формулы целого, булевского или строкового типа (например, в описании аналого-цифрового преобразователя).Если переменная, стоящая в левой части такой формулы, используется не только в дискретных действиях, но и в правых частях обычных уравнений, то эту формулу необходимо преобразовать в эквивалентную карту поведений. На Рис. 52 показана карта поведений для формулы вида $y = F(X)$

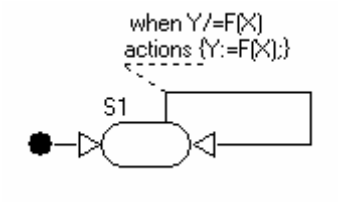


Рис. 52

Наконец, при использовании условных уравнений в неориентированных компонентах возможен случай, когда при переключении ветви условного выражения изменится структура уравнений и набор искомых переменных. В этом случае целесообразно также автоматически преобразовывать условное уравнение в эквивалентную карту поведений. На Рис. 53 показана карта поведения, соответствующая условному уравнению

$f_0(t,V) = if\ P\ then\ f_1(t,V)\ else\ f_2(t,V)$

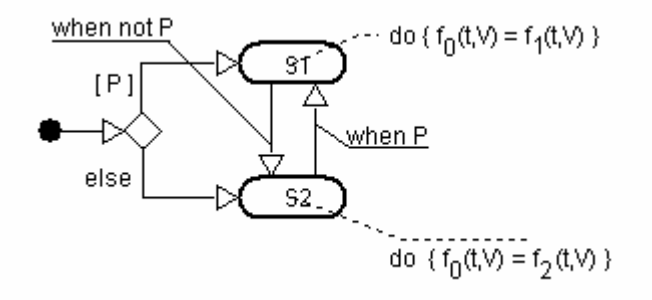


Рис. 53

Вложенному условному выражению будет соответствовать иерархическая карта состояний.

Использование пассивных объектов.

В сложных моделях возникает необходимость в более мощных средствах манипулирования данными, чем простые математические выражения. Кроме того, для описания логики мгновенных действий требуются алгоритмические операторы, такие как условный оператор, оператор цикла и т.д. В последовательных действиях также удобно использовать процедуры, а в выражениях функции. Наконец, в языках ООМ было бы правильно иметь возможность использовать помимо активных динамических объектов еще и обычные пассивные алгоритмические.

Разработчики инструментов моделирования используют при решении этой проблемы два подхода:

1) язык моделирования создается как расширение какого-нибудь языка программирования (Fortran, Simula, C, Java и т.п.);
2) язык моделирования поддерживает относительно небольшое множество алгоритмических конструкций, а для более сложных действий необходимо использовать внешние программные компоненты (DLL, COM и т.п.), написанные непосредственно на языках программирования.

Примерами первого подхода являются практически все ранние инструменты моделирования [29]. Примером языка ООМ, «надстроенного» над объектно-ориентированным языком программирования, является входной язык пакета AnyLogic [71], который является расширением языка Java. Примерами второго подхода являются такие пакеты как Omola, Dymola, а также пакеты семейства Model Vision [34], реализующие язык MVL. В последнем в качестве «внутреннего» алгоритмического языка используется небольшое подмножество языка программирования Ada, включающее в себя оператор присваивания, оператор вызова процедуры, условный оператор, оператор варианта, оператор цикла, оператор выхода из цикла и оператор возврата.

Первый подход помимо очевидных достоинств имеет и серьезные недостатки, которые являются оборотной стороной достоинств. Во-первых, сложный и мощный базовый язык программирования чрезвычайно затрудняет создание интерактивного инкрементального транслятора и заставляет ориентироваться на доступные пакетные компиляторы (например, `javac` для языка Java). Это означает, что контроль правильности модели откладывается до полной компиляции и возможны проблемы с диагностикой ошибок. Во-вторых, при использовании стандартного компилятора затруднен сам контроль семантики и некоторые действия приходится делать на стадии выполнения модели. Поэтому второй подход представляется более перспективным для языков ООМ. Процедуры и функции должны быть элементами описания поведения динамического объекта и не могут вызываться извне другими объектами, так как динамический объект взаимодействует с окружением только через внешние переменные. Внутри процедур и функций динамического объекта видимы переменные объекта (в теле процедуры они могут быть также изменены их значения). Для определения внутренних процедур и функций динамического объекта вполне достаточно относительно небольшого подмножества какого-нибудь известного языка программирования (например, Java). В то же время современные подходы к компонентному программированию (например, в MS .NET [43]) позволяют использовать в описании модели определения классов пассивных объектов, которые заданы в независимо разработанных с использованием любого удобного языка программирования «сборках».

Глава 5. Архитектура программных средств автоматизации моделирования сложных динамических систем.

В главе рассматривается и обосновывается архитектура инструментальных программных средств автоматизации системно-аналитического моделирования гибридных систем на примере пакета Model Vision Studium (MVS), поддерживающего язык MVL в качестве входного и удовлетворяющего требованиям, сформулированным во введении. Пакет MVS создан автором совместно с Д.Б.Иниховым и Ю.Б.Сениченковым [26]. Свободно распространяемую версию пакета можно найти, например, на образовательном сайте www.exponenta.ru (в разделе «Другие пакеты»).

Общая структура.

Пакет автоматизации моделирования (далее просто «пакет моделирования») должен решать следующие основные задачи:

- поддерживать интерфейс пользователя для создания математической модели исследуемой системы, а также обеспечивать контроль корректности этой модели;
- обеспечивать автоматическое построение компьютерной модели, соответствующей заданной математической;
- обеспечивать корректное проведение вычислительного эксперимента с компьютерной моделью на уровне абстракции математической модели.

Математическая модель исследуемой системы формулируется на входном языке MVL, базирующемся на формализме обобщенного гибридного автомата. Математическая модель как таковая является знаковой и непосредственно может интерпретироваться лишь человеком. Для того, чтобы математическая модель могла интерпретироваться компьютером, необходимо пре-

образовать ее в специальную программу – выполняемую модель -, содержащую объекты более низкого уровня абстракции.

Выполняемая модель включает в себя программу конкретной модели и исполняющую систему пакета моделирования (Рис. 54). Компьютерная модель включает в себя выполняемую модель в совокупности с операционной системой и аппаратной частью компьютера (Рис. 54). Компьютерная модель представляет собой уже некоторое физическое устройство, способное имитировать моделируемую систему в реальном мире. В работе [51] совокупность компьютера и моделирующей программы называется “электронным эквивалентом изучаемого объекта”.

Рис. 54

Именно в качестве физического устройства, т.е. имитатора изучаемой системы, компьютерная модель используется как для проведения автономных вычислительных экспериментов, так и при распределенном моделировании, в том числе в составе комплексного моделирующего стенда.

Для системно-аналитического моделирования необходимы выполняемые модели двух видов: 1) визуальная выполняемая модель; 2) «скрытая» выполняемая модель. В визуальную выполняемую модель помимо минимального ядра исполняющей системы включены средства поддержки визуализации результатов и активного вычислительного эксперимента (Рис. 55). Визуальная модель оформляется как самостоятельная выполняемая програм-

ма. Визуальная модель актуальна для отладки математической модели и демонстрации различных аспектов ее поведения.

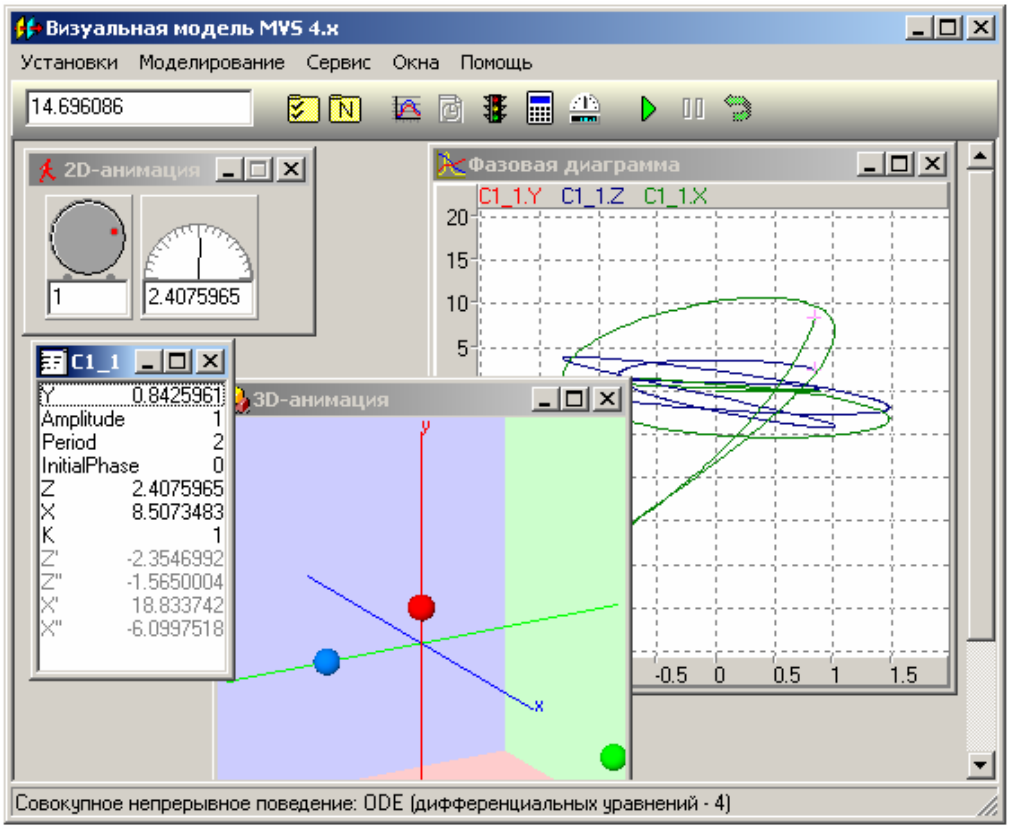


Рис. 55

«Скрытая» выполняемая модель не содержит никаких средств визуализации и оформляется как динамическая библиотека. «Скрытая» выполняемая модель актуальна для использования в составе других приложений, поддерживающих свою собственную визуализацию (например, в составе подсистемы оптимизации).

В принципе роль программы модели может играть интерпретатор математической модели. Такой подход имеет определенные достоинства и активно используется в ряде пакетов, однако для непрерывных моделей чрезвычайно неэффективен в части производительности и потому не может использоваться в MVS.

Каждому объекту математической модели (включая «неявные» объекты, такие как связи) в программе модели ставится в соответствие определенный программный объект в соответствии с синтаксисом объектно-

ориентированного языка программирования, выбранного в качестве языка реализации. В частном случае, когда математическая модель включает в себя лишь экземпляры стандартных объектов, для которых уже получен (автоматически или вручную) соответствующий программный код, то программа модели может свестись к небольшому объединяющему модулю, в котором создаются экземпляры требуемых программных объектов и устанавливаются взаимные ссылки. Такой подход широко используется, например, в пакетах «блочного моделирования». Если при этом еще и не требовать возможности отдельного от математической модели использования компьютерной модели, то эти объединяющие действия можно вполне интерпретировать в рамках редактора математической модели (примером может служить пакет SIMULINK в стандартном варианте). Язык MVL, однако, допускает определение пользователем своих собственных классов. Кроме того, даже при использовании только стандартных объектов использование свободной формы уравнений и ненаправленных связей требует сложных преобразований уравнений, которые зависят не только от поведения отдельных объектов, но и от соединений объектов. Поэтому в MVS программный код модели должен генерироваться заново для каждой конкретной модели.

Таким образом, минимальная конфигурация пакета моделирования гибридных систем должны включать в себя:

- средства редактирования математической модели;
- средства автоматической генерации программы модели;
- интегрированную среду, объединяющую все операции с математической и компьютерной моделью;
- исполняющую систему.

Эта минимальная конфигурация может быть расширена различными специальными подсистемами анализа и синтеза (например, параметрической оптимизации, анализа устойчивости линейных систем непрерывного управления классическими частотными методами, анализа временной устойчивости процессов логического управления, символического анализа [112] и т.д.).

Часть их этих подсистем может использовать непосредственно описание математической модели в каком-то согласованном представлении. Однако, самым общим случаем является представление исследуемой системы ее имитационной моделью. Например, в состав пакета MVS включена подсистема параметрической оптимизации, использующая скрытую выполняемую модель оптимизируемой системы.

Средства редактирования математической модели.

Математическая модель исследуемой системы формулируется на уровне абстракции языка моделирования MVL. Для пользователя основным является визуальное представление модели, в котором элементы описания (уравнения, карты поведений, структурные схемы) представляются в естественной графической форме. Для этого используются специальные редакторы: редактор уравнений (Рис. 58), разработанный С.Толстинским,

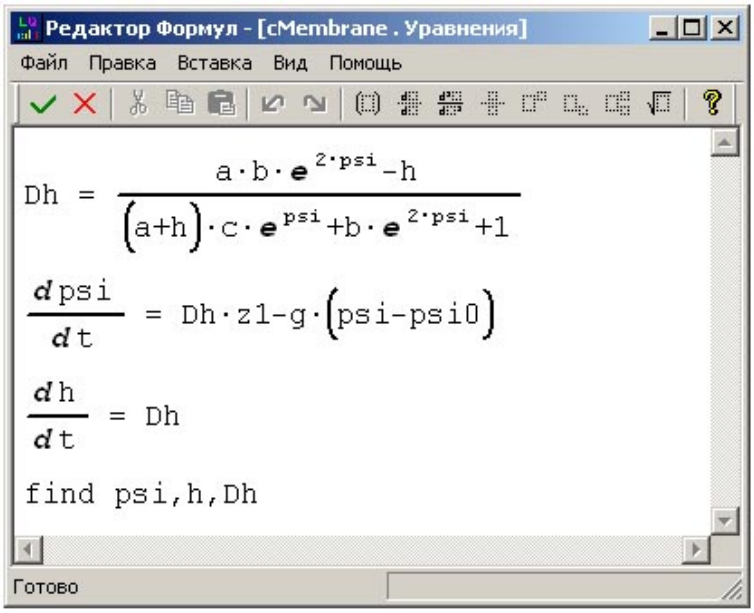


Рис. 56

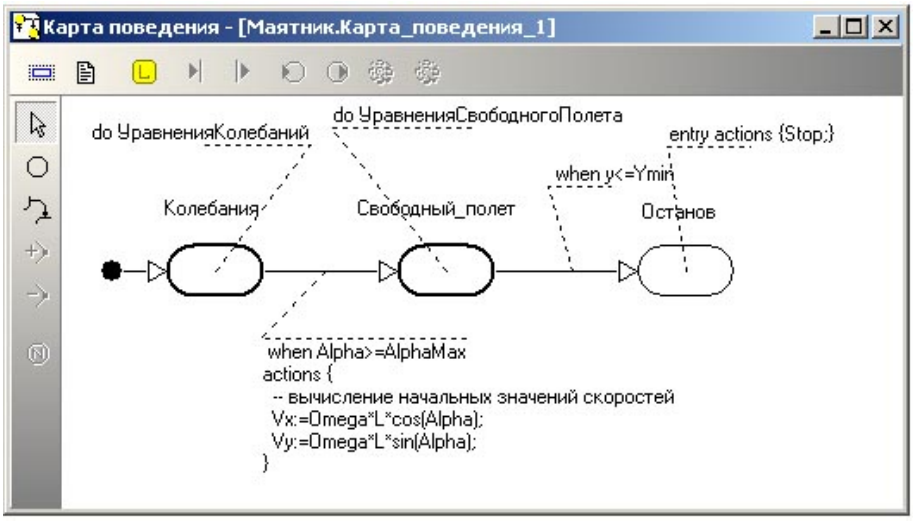


Рис. 57

редактор карты поведений (Рис. 57), редактор структурной схемы (Рис. 58).

В ряде случаев, однако, оказывается более удобным работать с текстовым представлением модели непосредственно на языке MVL. В текстовом представлении информация о положении графических элементов задается с помощью специального оператора `pragma`.

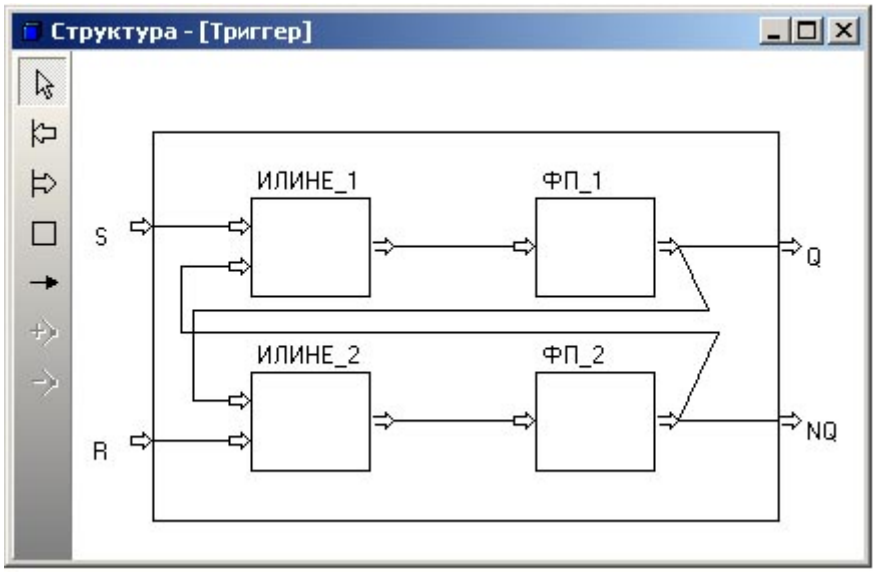


Рис. 58

Известно, что при работе в визуальных средах наиболее удобным для пользователя является использование т.н. инкрементного (пошагового) транслятора [90]. Использование инкрементного транслятора позволяет не-

медленно контролировать правильность вводимой пользователем языковой конструкции (например, системы уравнений или последовательности действий в переходе) в контексте ее окружения. Однако, для использования инкрементного транслятора необходимо наличие некоторого внутреннего структурированного представления информации о математической модели.

Принципиальной особенностью архитектуры пакета MVS является использование внутреннего структурированного объектного представления математической модели в качестве основного. Внутреннее представление включает в себя совокупность взаимосвязанных мета-объектов, отражающих содержание и структуру математической модели на уровне абстракции языка MVL. Объекты внутреннего представления рассматриваются как долговременные (persistent) и сохраняются вместе со связями в объектно-ориентированной базе данных `mvBase`, разработанной автором [32]. В этой базе данных в качестве предопределенного поддерживается мета-класс «математическое выражение», позволяющий хранить разобранные выражения в виде дерева. Схема данных базы данных проекта соответствует грамматике языка MVL. Внутреннее представление является единственной хранимой информацией о проекте (Рис. 59). Внешние представления – визуальное и текстовое – автоматически воссоздаются по внутреннему представлению при необходимости (например, открытии окна редактирования соответствующего объекта). Естественная структура данных внутреннего представления позволяет воссоздавать внешние представления, проверять контекст окружения и проводить автоматические преобразования (например, символьное дифференцирование) достаточно быстро. Информация об импортируемых пакетах также берется из их внутреннего представления. При редактировании проекта может быть открыто одновременно несколько баз данных, но из них только одна – проект - на запись.

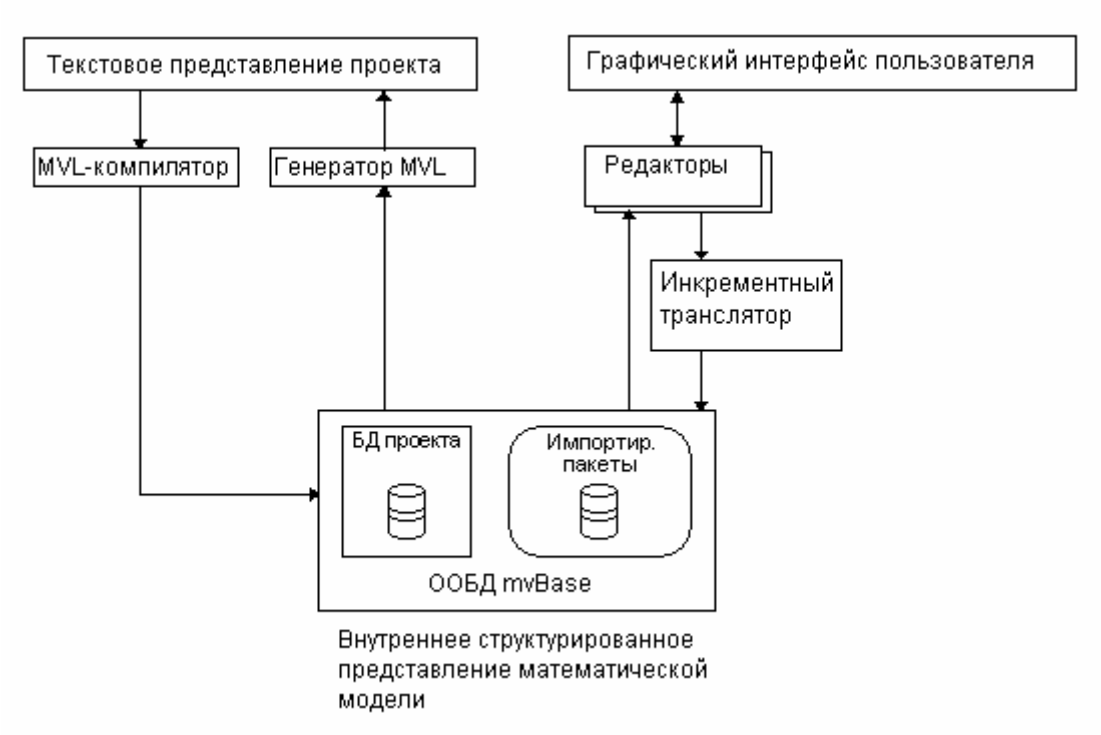


Рис. 59

Инкрементный транслятор преобразует законченную конструкцию MVL в соответствующий объект внутреннего представления, учитывая контекст окружения. В тех случаях, когда в результате редактирования изменяется интерфейс активного объекта, то проводится проверка корректности его окружения на следующем уровне вложенности (например, если изменяется сигнатура алгоритмической функции класса, то проверяется корректность класса, если изменяется определение внешней переменной класса, проверяется корректность проекта в целом). Процесс трансляции выступает в качестве транзакции ООБД. В случае успешного завершения трансляции соответствующая транзакция фиксируется и все изменения сохраняются в БД, в случае обнаружения ошибки выводится диагностика и транзакция откатывается. Наличие структурированного внутреннего представления позволят также наделять специализированные редакторы визуального представления некоторыми «интеллектуальными» чертами: например, редактор структурных схем просто не позволяет соединить связью внешние переменные, не совместимые по типу.

Средства генерации программы модели.

Выполняемая модель соответствует экземпляру класса `Model`. Таким образом, для класса `Model`, его локальных классов, их элементов, суперкласса, а также других активных динамических объектов, используемых в структурной схеме класса `Model`, должны быть сгенерированы определения соответствующих программных объектов на некотором объектно-ориентированном языке программирования. В качестве промежуточного языка программирования в пакете MVS выбран язык `Object Pascal` .(одной из причин выбора этого языка была очень высокая скорость работы компилятора `dcc32`). Эти программные объекты в основном являются потомками некоторых базовых классов объектов периода выполнения, определенных в модулях исполняющей системы.

Генератор «кода» формирует (Рис. 60) отдельный модуль `u_xxx.pas` для каждого активного объекта (в том числе и импортируемых), модуль `common.pas` для констант и алгоритмических функций проекта, а также констант и алгоритмических функций из импортируемых пакетов, и головной модуль `model.dpr`. В головном модуле определяется тип выполняемой модели – визуальная или скрытая. Далее с помощью компилятора командной строки `Object Pascal` сгенерированные модули компилируются и объединяются с модулями исполняющей системы, содержащими определения базовых объектов (Рис. 60). Поскольку перед генерацией кода проводится полный контроль корректности класса `Model`, ошибки при компиляции могут быть вызваны только ошибками в пакете моделирования.

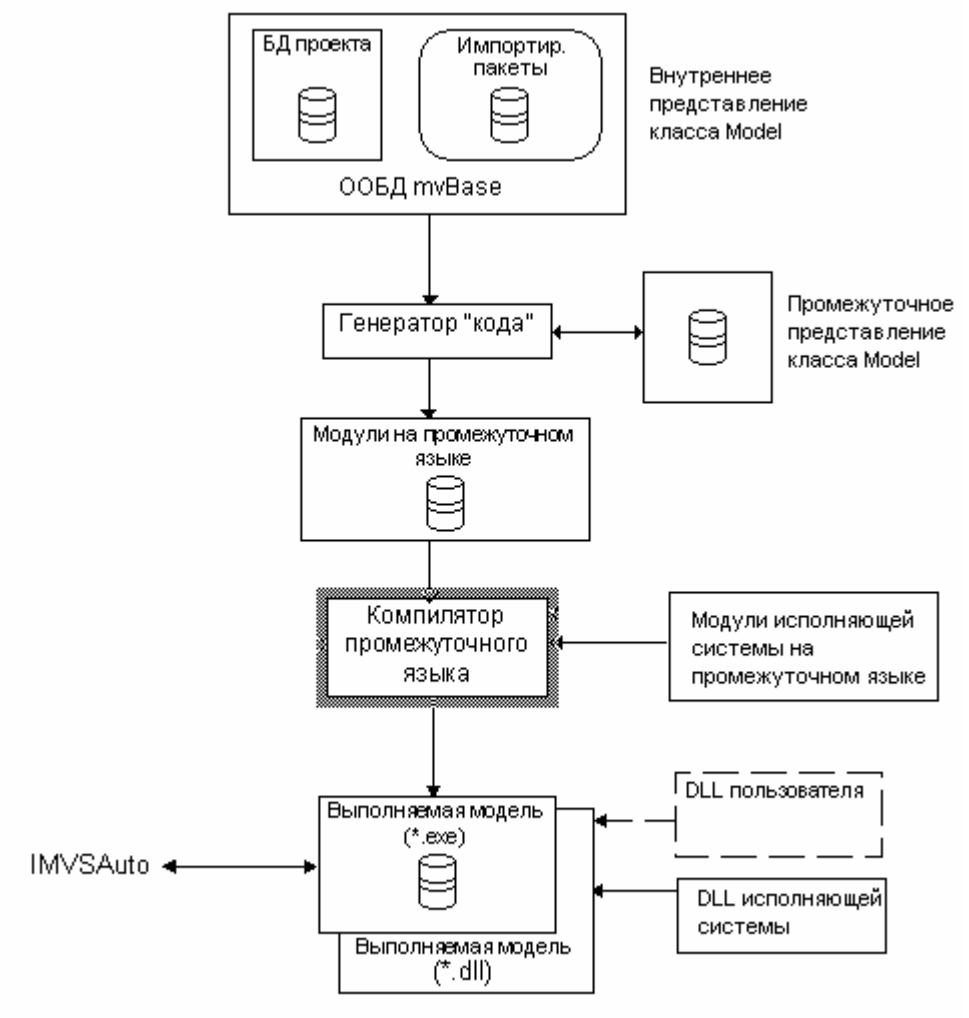


Рис. 60

В общем случае программный код формируется в соответствии не с исходным внутренним представлением, а с некоторым промежуточным, формируемым в процессе генерации (Рис. 60). Необходимость промежуточного внутреннего представления модели связана прежде всего с преобразованиями уравнений. Глубина преобразований определяется двумя факторами:

- входит ли в состав выполняемой модели хотя бы один гибридный активный объект (т.е., объект с непустой картой поведений);
- используются ли в выполняемой модели ненаправленные связи.

В случае, если выполняемая модель является чисто непрерывной, можно уже на стадии генерации кода сформировать полную систему уравнений всей модели, провести ее анализ, определить искомые переменные и сделать необходимые преобразования (разрыв алгебраических циклов, символьное

дифференцирование, символьное разрешение алгебраических уравнений). В ходе этих преобразований может сильно измениться представление уравнений (в том числе и для импортируемых классов).

В случае, если гибридные объекты имеются, но в модели используются только направленные связи, выполняется анализ и преобразование отдельных систем уравнений. Анализ полной системы уравнений модели может быть выполнен только на стадии исполнения модели после каждого переключения. При наличии гибридных объектов в «код» модели при генерации встраивается дополнительная служебная информация о переменных, участвующих в уравнениях, которая будет необходима при анализе.

Наконец, в случае наличия гибридных объектов и использования ненаправленных связей на стадии генерации вообще ничего сказать о совокупной системе уравнений нельзя.

При выполнении гибридной модели анализ новой совокупной системы уравнений проводится после каждого переключения. В результате анализа может выявиться необходимость в символьном дифференцировании некоторых уравнений. В этом случае возможен возврат к генерации кода на основе сохраненного промежуточного внутреннего представления.

Интегрированная среда.

Интегрированная среда пакета MVS объединяет все составляющие пакета в единый инструмент, поддерживающий все основные операции стандартного цикла разработки и отладки модели (Рис. 61). В рамках интегрированной среды пользователь имеет возможность:

- редактировать математическую модель в интерактивном режиме с использованием инкрементным транслятором (Рис. 61);
- создать файл текстового представления математической модели на языке MVL и редактировать его в любом текстовом редакторе;
- импортировать математическую модель из текстового представления;

- проверить корректность всего проекта в целом.;
- создать выполняемую модель (визуальную или скрытую) и при необходимости сохранить ее как независимый программный модуль;
- запустить визуальную выполняемую модель;
- прекратить выполнение визуальной модели;
- выполнить параметрическую оптимизацию математической модели.

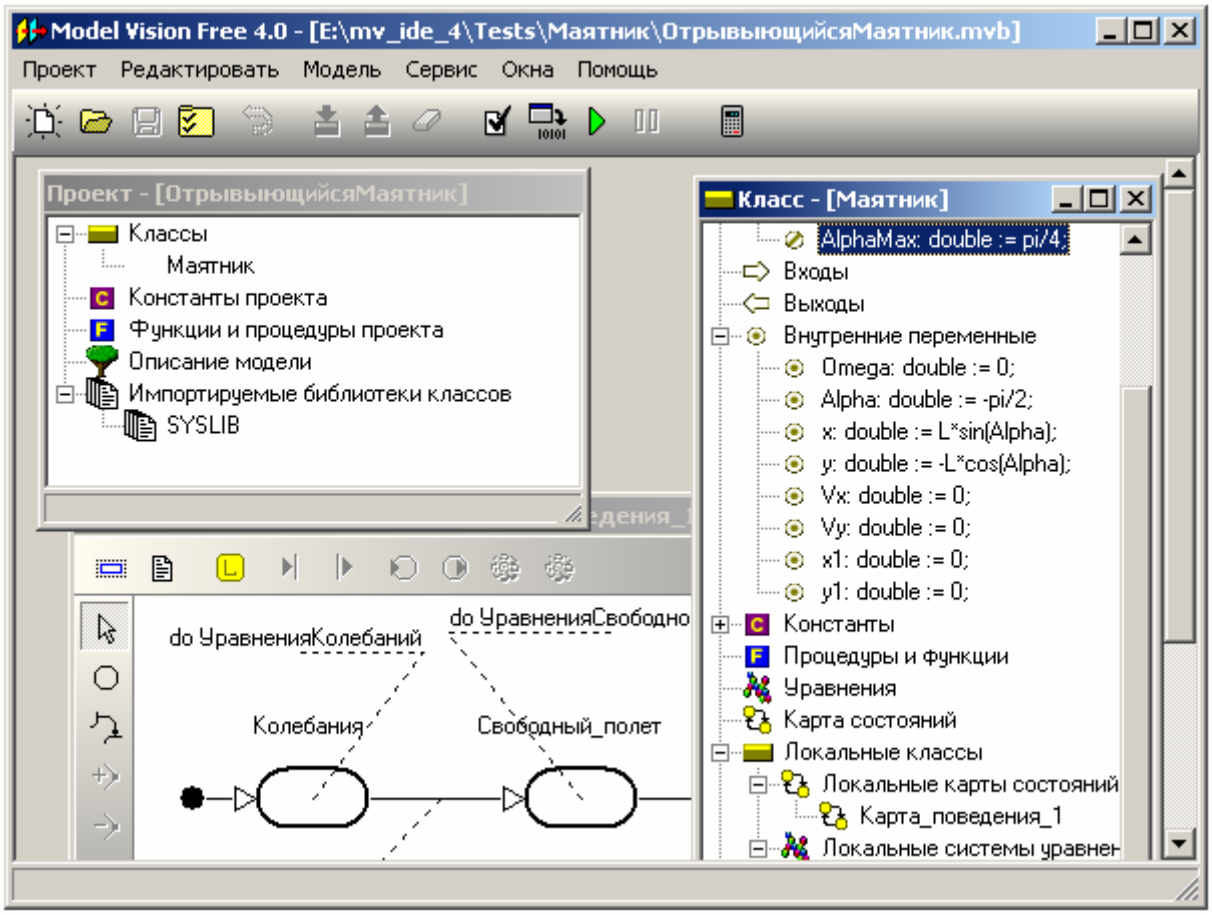


Рис. 61

В интегрированную среду пакета могут быть встроены и другие специальные подсистемы анализа и синтеза гибридных моделей.

Исполняющая система.

Исполняющая система пакета моделирования MVS включает в себя:

- определения базовых классов для программных объектов модели;
- численные библиотеки;

- блок продвижения модельного времени;
- средства поддержки активного вычислительного эксперимента..

Определения базовых классов.

Базовые классы исполняющей системы определяют обобщенные элементы выполняемой модели (активный объект, переменная, уравнение, система уравнений, карта поведений, состояние, переход и т.д.). При генерации программы модели создаются определения конкретных программных классов, которые соответствуют классам, определяемым в проекте и являются потомками базовых классов исполняющей системы. Определения базовых классов исполняющей системы используются при компиляции модулей программы модели на промежуточном языке (Рис. 60).

Численные библиотеки.

В настоящее время практически все известные численные методы ориентированны на возможно более быстрое нахождение значений искомых переменных в определенные моменты времени для чисто непрерывных систем, задаваемых системой дифференциальных, алгебраических или дифференциально-алгебраических уравнений [59,72,33]. Имеются также специальные методы для уравнений задержки [65,66]. Непосредственно эти методы не могут быть использованы для моделирования гибридных систем. Даже для простейшей гибридной системы, задаваемой фиксированной системой уравнений с изменяющимися правыми частями, непосредственное использование стандартных численных методов может приводить к значительным ошибкам.

Как было отмечено в главе 1, при моделировании гибридных систем возможны два подхода к использованию численных методов:

1) создание на базе стандартных численных методов специальных численных методов для решения гибридных задач;
2) создание специальной надстройки над стандартными численными методами, которая занималась бы дискретными аспектами решаемой задачи.

По существу различие между двумя подходами заключается в том, доступны или нет в надстройке (она необходима в обоих случаях) внутренние данные стандартного численного метода. В первом подходе они доступны, а во втором стандартный численный метод используется как «черный ящик». Использование внутренних данных численного метода может чрезвычайно упростить решение многих задач гибридного моделирования (например, поиск точки переключения). В то же время, структура внутренних данных в современных программных реализациях стандартных численных методов столь разнородна, что в рамках первого подхода трудно говорить об использовании множества различных методов для эффективного решения конкретных задач. Разработка же собственных реализаций влечет за собой появление ошибок. Поэтому при создании пакета MVS было принято решение использовать второй подход. Использованы готовые программные реализации стандартных методов, а в качестве надстройки выступает блок продвижения модельного времени.

Численные библиотеки пакета MVS оформлены в виде отдельной динамической библиотеки `_mathmvs.dll`. Эта динамическая библиотека написана на языке Фортран и содержит программные реализации всех элементарных функций, включенных в язык MVL, некоторых внутренних математических функций, а также численных методов решения систем дифференциальных, алгебраических и дифференциально-алгебраических уравнений. Идеология языка MVL такова, что для вычисления фазовой траектории модели на участках непрерывности не требуется каких-то специальных численных методов и можно использовать существующие хорошо апробированные численные методы, предназначенные для непрерывных систем (с некоторой неизбежной адаптацией программных интерфейсов). Для каждой группы уравнений реализован метод – «автомат», используемый по умолчанию, и некоторый набор специализированных методов. Для каждого типа уравнений пользователь может выбрать метод решения. «Автомат» предназначен для автоматического выбора наиболее походящего численного метода для реше-

ния данной конкретной задачи в данной точке фазовой траектории. «Автомат» предоставляет пользователю максимальную информацию о встреченных трудностях. Анализируя эти сообщения, пользователь может выбрать наиболее эффективный для своей задачи специализированный метод. «Автомат» предпочтительно использовать на начальной стадии отладки модели, когда ее численные свойства еще не вполне ясны пользователю, а затем следует выбрать подходящий специализированный метод.. Специализированные методы предназначены для эффективного решения задач определенного типа (нежестких, жестких, колебательных и т.п). Ядро исполняющей системы в начальной точке и затем после каждого переключения определяет тип совокупной системы уравнений и использует выбранный пользователем для данного типа системы уравнений численный метод.

Блок продвижения модельного времени.

Блок продвижения модельного времени является ядром исполняющей системы и обеспечивает интерпретацию гибридных карт поведений, а также вызов численных методов для вычисления точек фазовой траектории модели. Непрерывная составляющая модельного времени является независимым процессом, глобальным для всех составляющих модели и единственным источником движения вдоль фазовой траектории. Поэтому блок продвижения модельного времени часто называют «`model engine`» [80,101]. Для гибридного модельного времени в ядре следует выделить блок продвижения дискретной составляющей модельного времени («`discrete model engine`» - DME) и блок продвижения непрерывной составляющей модельного времени («`continuous model engine`» - CME). Блок продвижения времени является специальной надстройкой над классическими численными методами, которая обеспечивает решение специфических проблем гибридного моделирования.

Блок продвижения модельного времени можно также рассматривать как интерпретатор эквивалентного последовательного процесса модели, по-

строенного по правилам синхронного объединения гибридных автоматов, рассмотренным в Главе 2. Эквивалентный последовательный процесс и совокупная глобальная система уравнений строятся динамически во время исполнения модели. Следует отметить, что этот эквивалентный процесс объединяет не только карты поведений активных объектов модели, но также и ряд «служебных» процессов самой исполняющей системы. К «служебным» процессам относятся:

- процессы обновления диаграмм;
- процесс синхронизации с реальным временем;
- процесс останова по условию.

Таким образом, эквивалентный последовательный процесс существует даже для чисто непрерывной модели. Ядро исполняющей системы осуществляет динамическое построение и интерпретацию эквивалентного процесса по единому алгоритму, не различая прикладных или «служебных» процессов.

В выполняемой модели ядро исполняющей системы реализуется как один независимый параллельный поток управления (`thread`). Таким образом, в визуальной модели одновременно выполняются поток главного окна и поток ядра, а в «скрытой» модели» поток вызывающего приложения и поток ядра. Это делается для того, чтобы избежать «подвисания» модели во время трудоемких вычислений. Отображение информации в визуальной модели должно проводиться исключительно лишь посредством сообщений, посылаемых соответствующим окнам, или через специальные синхронизированные процедуры.

Следует отметить, что в визуальной модели в качестве дополнительного неявного «потока управления» выступает также и пользователь, который может интерактивно вмешиваться в ход вычислительного эксперимента и изменять состояние модели

Алгоритм продвижения гибридного модельного времени.

На Рис. 62 приведен укрупненный алгоритм продвижения гибридного модельного времени модели. «Горизонтальная» цепочка переходов отражает продвижение дискретной составляющей модельного времени во «временной щели», а «вертикальная» цепочка отражает продвижение непрерывной составляющей модельного времени. Все действия выполняются последовательно в реальном времени в потоке управления ядра исполняющей системы. В результате выполнения этого алгоритма имитируются параллельные процессы, протекающие в модельном времени.

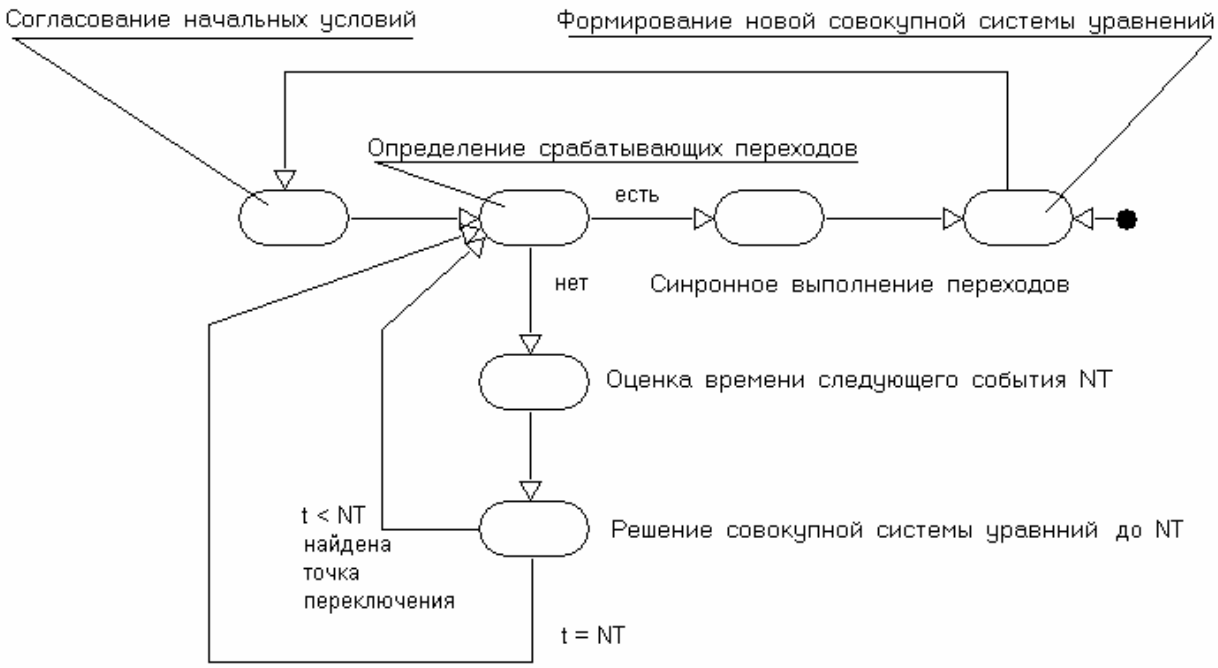


Рис. 62

Этот алгоритм является развитием классического алгоритма продвижения времени «по событиям» в непрерывно-дискретных моделях [10,50,16] применительно к картам поведений и активному вычислительному эксперименту.

В ядре исполняющей системы имеется четыре специальных списка:

- КАЛЕНДАРЬ (по аналогии с [16]) – список переходов с условием `after`, упорядоченный по значению времени срабатывания;

- СПИСОК_СРАБАТЫВАЮЩИХ_ПЕРЕХОДОВ – список переходов, которые готовы к срабатыванию в данной точке дискретного времени;
- СПИСОК_ОЖИДАЮЩИХ_ПЕРЕХОДОВ – список переходов, ожидающих истинности логического предиката (условие срабатывания `when`). Этот список задает некоторый логический предикат $P(V,t) = \bigcup_{i=1..n} P_i(V,t)$, где V - множество переменных модели, $P_i(V,t)$ - условие срабатывания i-го перехода, n - число переходов в списке;
- СПИСОК_УРАВНЕНИЙ – список уравнений, составляющих в данный момент гибридного времени совокупную систему уравнений модели. С этим списком также соотносится множество искомых переменных $v \subseteq V$. В общем случае совокупная система уравнений может находиться в двух формах:

  1) исходной, когда она представляет собой «механическую» сумму уравнений, не преобразованных к вычислимой форме;

  2) вычислимой, когда исходная совокупность уравнений проанализирована и преобразована к вычислимой форме.

  Любое добавление или изъятие уравнений переводит совокупную систему уравнений из вычислимой формы в исходную. Для чисто непрерывной модели совокупная система уравнений генерируется уже в вычислимой форме.

Создание и уничтожение активного объекта.

Экземпляр активного объекта создается при создании экземпляра охватывающего активного объекта, при выполнении оператора `new` в последовательности мгновенных действий, а также при входе в состояние, в котором этот объект является деятельностью. Экземпляр класса Model создается исполняющей системой автоматически при каждом запуске модели. При создании экземпляра активного объекта:

- его система уравнений добавляется к совокупной системе уравнений модели;
- инициализируется начальное состояние карты поведений, если она не пуста;
- создаются экземпляры статических локальных активных объектов, если данный объект имеет непустую внутреннюю структуру.

Экземпляр активного объекта уничтожается при уничтожении экземпляра охватывающего активного объекта, при выполнении оператора `destroy` в последовательности мгновенных действий, а также при выходе из состояния, в котором этот объект является деятельностью. Экземпляр класса Model уничтожается исполняющей системой автоматически по окончании прогона модели. При уничтожении экземпляра активного объекта:

- его система уравнений удаляется из совокупной системе уравнений модели;
- пассивизируется текущее состояние карты поведений;
- уничтожаются экземпляры статических локальных активных объектов, если данный объект имеет непустую внутреннюю структуру.

Инициализация и пассивизация состояния в карте поведений.

При инициализации нового состояния карты поведений:

- выполняется последовательность входных действий;
- если состоянию приписан некоторый активный объект, то создается соответствующий экземпляр;
- проводится анализ исходящих переходов: безусловный переход (он может быть только один) заносится в СПИСОК_СРАБАТЫВАЮЩИХ_ПЕРЕХОДОВ, переходы, зависящие от времени (условие `after`) заносятся в КАЛЕНДАРЬ, переходы, зависящие от дискретного события (условие `when`) заносятся в СПИСОК_ОЖИДАЮЩИХ_ПЕРЕХОДОВ.

При пассивизации текущего состояния карты поведений:

- уничтожается экземпляр активного объекта, приписанного этому состоянию;
- ссылки на исходящие переходы удаляются их всех системных списков;
- выполняется последовательность выходных действий.

Выполнение перехода.

При выполнении перехода:

- пассивизируется исходное состояние перехода;
- выполняется последовательность действий перехода.
- активизируется конечное состояние перехода;

В случае, если последовательность действий содержит оператор посылки сигнала, то этот сигнал немедленно передается по связям на входы других компонент. После этого в СПИСКЕ_ОЖИДАЮЩИХ_ПЕРЕХОДОВ ищутся переходы, ожидающие соответствующих сигналов. Если среди найденных переходов имеются такие, у которых еще и выполняется (или отсутствует) охраняющее условие `guard`, то они переносятся в СПИСОК_СРАБАТЫВАЮЩИХ_ПЕРЕХОДОВ.

Продвижение дискретной составляющей времени.

При каждом запуске выполняемой модели создается экземпляр класса Model. При этом инициализируются начальные состояния карт поведений активных объектов, входящих в модель, а их системы уравнений добавляются к совокупной системе уравнений модели. Кроме того, инициализируются начальные состояния «служебных» процессов исполняющей системы. Таким образом, в начале каждого прогона формируется начальное состояние эквивалентного последовательного процесса и проводится начальное заполнение совокупной системы уравнений Q.

На Рис. 63 показана карта состояний для алгоритма продвижения гибридного времени.

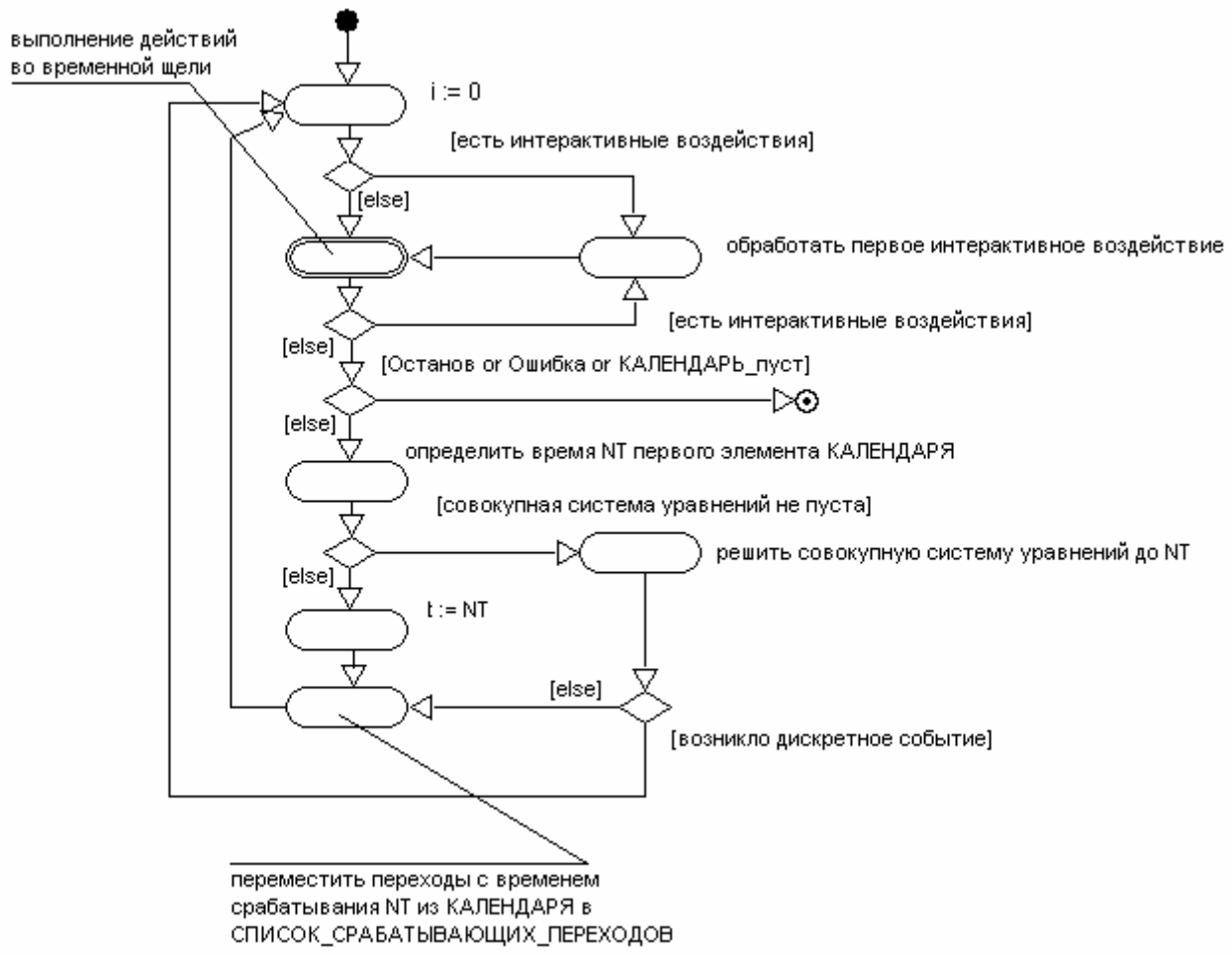


Рис. 63

Вложенная карта состояний «Выполнение действий во временной щели» показана на Рис. 64. Эта карта состояний соответствует общему случаю алгоритму продвижения дискретного времени во временной щели для гибридной модели.

Если совокупная система уравнений модели Q была изменена с момента предыдущего анализа, то выполняется новый анализ и, возможно, ее преобразование к численно решаемой системе Q_N согласно алгоритму, приведенному в главе 2.

На следующем шаге ищется численное решение алгебраической составляющей полученной совокупной системы уравнений с целью определения согласованных начальных значений искомых переменных в данной точке гибридного времени. После нахождения согласованных значений дискретная составляющая i гибридного времени h увеличивается на единицу (Рис. 64).

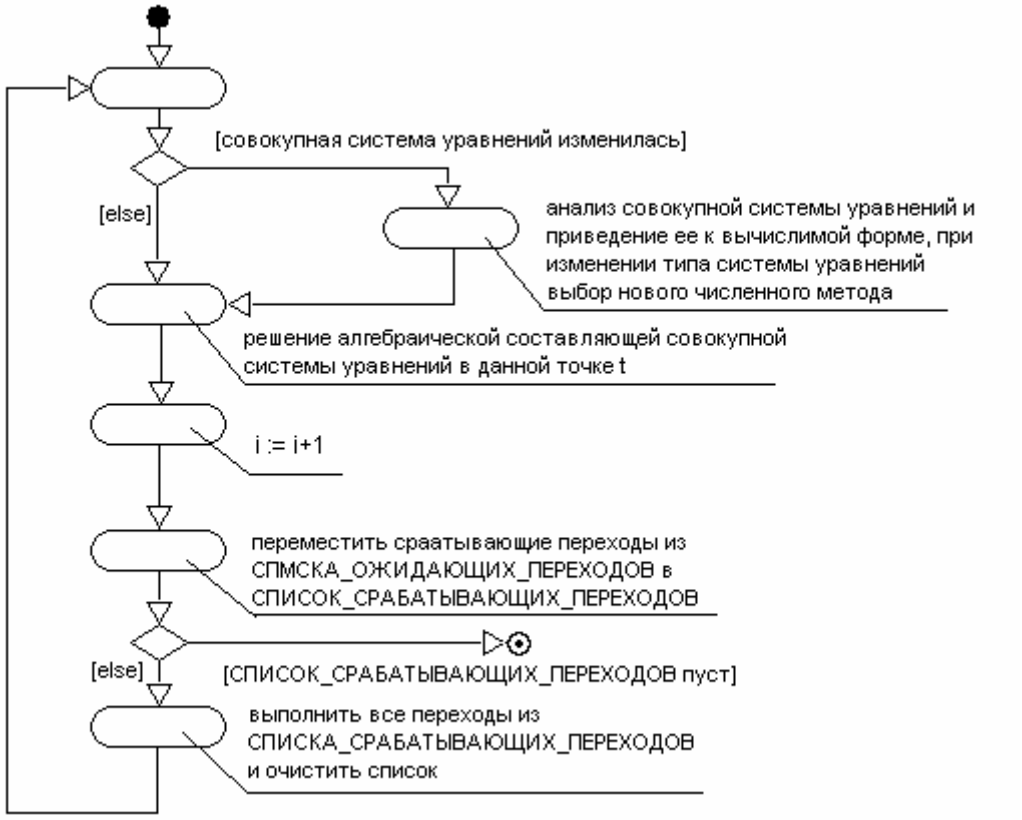


Рис. 64

На основе полученных согласованных значений искомых переменных проверяются условия срабатывания переходов в СПИСКЕ_ОЖИДАЮЩИХ_ПЕРЕХОДОВ и готовые к срабатыванию переносятся в СПИСОК_СРАБАТЫВАЮЩИХ_ ПЕРЕХОДОВ.

Если СПИСОК_СРАБАТЫВАЮЩИХ_ПЕРЕХОДОВ не пуст, то выполняются все переходы из этого списка и список очищается. Отметим, что порядок выполнения переходов не имеет значения, поскольку язык MVL допускает только синхронное объединение гибридных автоматов. При этом предполагается, что: 1) каждый компонент содержит только одну карту поведений (возможно, иерархическую); 2) взаимодействие компонент осуществляется только через уравнения связи (за исключением сигналов). Поэтому при выполнении перехода результаты выполнения других переходов в данный момент дискретного времени никак не могут на него влиять. На результат выполнения влияет только причинно-следственная цепочка мгновенных

действий в самом компоненте (выходные действия в исходном состоянии → действия перехода → входные действия в конечном состоянии).

В результате выполнения переходов в общем случае изменятся значения переменных, изменится состав активных объектов и изменится совокупная система уравнений. Далее указанный цикл будет повторяться до тех пор, пока СПИСОК_СРАБАТЫВАЮЩИХ_ ПЕРЕХОДОВ не окажется пуст. Это означает, что данная временная щель закончилась, в дискретной составляющей модели нет больше «источников движения» и для дальнейшего продвижения модели необходимы либо внешние воздействия, либо продвижение непрерывной составляющей времени. При выполнении мгновенных действий во временной щели контролируется поведение Зенона: если число срабатываний любого перехода в данной щели превышает некоторое значение, то выдается предупреждающее сообщение.

В случае, если имеются внешние интерактивные воздействия, обрабатывается первое в очереди воздействие и повторяется цикл обработки мгновенных действий, то есть внешнее воздействие может инициировать следующую временную щель, «склеенную» в непрерывном времени с предыдущей. Если таких воздействий нет, то можно продвигать непрерывную составляющую времени. Для этого по первому элементу КАЛЕНДАРЯ прогнозируется время (непрерывная составляющая) следующего дискретного события NT. В визуальной модели такой элемент всегда существует (как минимум, это переход в процессе синхронизации с реальным временем). В «скрытой» модели в случае, когда КАЛЕНДАРЬ пуст, значение NT либо определяется вызывающим приложением, либо в качестве него используется какое-нибудь достаточно большое число, близкое к предельному представимому в разрядной сетке компьютера. Если разность $NT - t$ много раз подряд оказывается близкой к минимальному представимому в разрядной сетке положительному числу, то выдается предупреждающее сообщение о возможном парадоксальном поведении второго типа.

Если совокупная система уравнений пуста, то есть в данной точке модель чисто дискретная, то непрерывной составляющей модельного времени просто присваивается значение NT и все элементы КАЛЕНДАРЯ с временем срабатывания NT переносятся в СПИСОК_СРАБАТЫВАЮЩИХ_ПЕРЕХОДОВ (Рис. 63). В противном случае с помощью СМЕ выполняется решение совокупной системы уравнений на непрерывном интервале $[t, NT]$. Это решение может завершиться либо успешным достижением момента NT, либо в какой-то момент $t^* < NT$ возникает дискретное событие. Этим событием может быть выполнение условий срабатывания переходов, переключение ветвей условного уравнения, внешнее воздействие или выдача пользователем команды «Останов». В первом случае все элементы КАЛЕНДАРЯ с временем срабатывания NT переносятся в СПИСОК_СРАБАТЫВАЮЩИХ_ПЕРЕХОДОВ (Рис. 63).

Продвижение непрерывной составляющей времени.

Задачей блока продвижения непрерывной составляющей модельного времени СМЕ является получение с заданной точностью значений искомых переменных совокупной системы уравнений для момента времени NT (Рис. 65). В случае, когда в результате добавления или удаления уравнений изменяется тип системы Q_N (например, система уравнений из чисто дифференциальной стала дифференциально-алгебраической), необходимо заменить численный «решатель».

Для этого полученная в результате анализа система уравнений Q_N из объектной формы преобразуется к некоторой вспомогательной скалярной форме Q_N^S, которая вместе с начальными значениями передается численному методу. Если во временной щели в модели изменялись значения переменных или совокупная система уравнений, то численный метод «переразгоняется», то есть заново инициализирует свои рабочие данные. Кроме того, численному методу передаются две `callback`-ссылки: ссылка на процедуру вычисления правых частей уравнений и ссылка на процедуру анализа результатов

очередного шага интегрирования. Процедура вычисления правых частей уравнений преобразует текущие значения искомых величин из вспомогательной формы в исходную объектную, временно присваивает непрерывной составляющей модельного времени требуемое промежуточное значение, вычисляет значения правых частей и преобразует эти значения во вспомогательную форму. Такие преобразования неизбежны при использовании современных численных методов общего назначения, не ориентированных на работу с объектными приложениями.

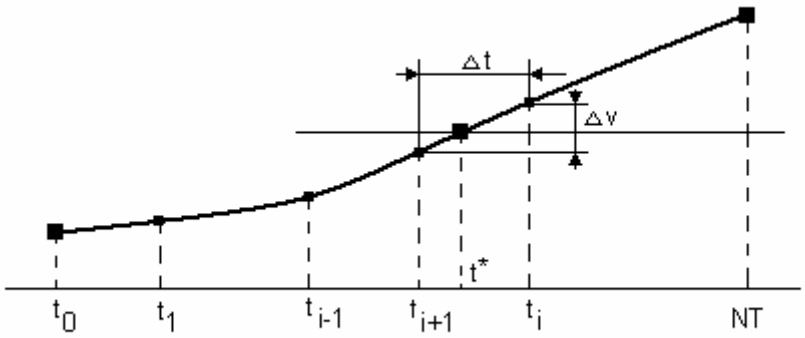


Рис. 65

Возможны два варианта дальнейших действий:

1) численный метод благополучно вычисляет значение $v(NT)$. В общем случае на интервале $[t_0, NT]$ численный метод сделает несколько шагов (величину шага методы в основном выбирают автоматически), вычислив промежуточные значения переменных $v(t_1), v(t_2), ..., v(t_i)$ (Рис. 65). СМЕ присвоит непрерывной составляющей модельного времени значение NT и вернет управление DME. Последний переместит первый элемент КАЛЕНДАРЯ в СПИСОК_ПЕРЕХОДОВ и начнет обработку временной щели по указанному выше алгоритму.

2) Зависимость $v(t)$ такова, что в некоторый момент $t^* \mid t_0 < t^* \le NT$ предикат $P(v^*, t^*)$ становится истинным (Рис. 65). Это означает, что в момент t^* в модели возникает дискретное событие, обусловленное изменением непрерывных переменных модели v (остальные переменные могут изменять свое зна-

чение только во временной щели). В этом случае процесс вычисления $v(t)$ должен быть прерван и управление возвращено DME, который начнет обработку временной щели. Для этого необходимо определить с заданной точностью момент t^* и значения переменных $v(t^*)$. Эту задачу будем называть задачей поиска точки переключения.

Поиск точки переключения.

Прежде всего, можно выделить важный частный случай, когда логические предикаты $P_i(V,t)$ можно привести к виду $G_i(V,t)=0$, где $G_i(V,t)$ - некоторая непрерывная по всем переменным функция. В этом случае можно использовать некоторые специальные численные методы, которые позволяются вместе с решением системы Q_N^S находить ближайший корень дополнительной системы алгебраических уравнений $G(V,t)=0$ [36]. Чтобы обеспечить такую возможность, генератор кода анализирует условия срабатывания переходов и для предикатов, приводимых к указанному выше виду, генерирует специальный программный код, обеспечивающий возможность вычисления значения $G_i(V,t)$. СМЕ ищет точку переключения таким способом, если все ожидающие предикаты приводимы к алгебраическим уравнениям. При использовании этого способа процедура анализа результатов шага вызывается только в визуальной модели и только для контроля интерактивных воздействий пользователя.

В случае, когда все предикаты $P_i(V,t)$ или часть из них являются логическими выражениями общего вида, СМЕ реализует внешний по отношению к численному методу алгоритм поиска точки переключения. Для этого пи вызове численного метода активизируется обращение к процедуре анализа результатов очередного шага и запоминается значение $v(t_0)$. По завершении каждого шага интегрирования в момент t_i численный метод вызывает процедуру анализа результатов шага, которая вычисляет значение предиката $P(V,t_i)$. Если предикат ложен, то значение $v(t_i)$ запоминается и решение про-

должается. Если предикат истинен, то решение прекращается и данная точка запоминается как $t_T = t_i, t_F = t_{i-1,}$. Далее СМЕ начинает выполнять поиск по следующему алгоритму:

1) $t := t_F, v := v(t_F), t_x := t_F + k \cdot (t_T - t_F)$;

2) ищется решение $v(t_x)$ (по определению численный метод не будет делать промежуточных шагов на этом интервале);

3) если $P(V(t_x), t_x) = true$ то $t_T := t_x$, иначе $t_F := t_x$;

4) если не выполнены условия точности, то идти к (1).

В зависимости от значения коэффициента k это может быть метод половинного деления, «золотого сечения» и т.п.

Цикл завершается успешно, когда выполнены условия требуемой точности по всем искомым переменным

$$0.9 \cdot \Delta v_i \le \hat{v}_i \cdot \delta + \Delta \quad | \quad i = 1..n,$$

где $\Delta v_i = |v_i(t_T) - v(t_F)|$ $\hat{v}_i = \dfrac{|v_i(t_T) + v_i(t_F)|}{2}$, δ - заданная относительная погрешность решения, Δ - заданная абсолютная погрешность решения и по времени

$$0.9 \cdot |t_T - t_F| \le t_T \cdot \delta + \Delta$$

В качестве приближенного решения принимается t_T, $v(t_T)$.

Цикл завершается ошибкой, если

$$|t_T - t_F| \le t_T \cdot \delta_{\min},$$

где $\delta_{\min}$ - некоторая минимальная относительная погрешность, допустимая в разрядной сетке компьютера.

Полученное по методу деления приближенное решение t_T всегда находится «справа» от истинной точки переключения t^*. Между тем в некоторых случаях сама система уравнений Q_N может быть не определена за точкой переключения. Например, для системы уравнений $\dfrac{dX}{dt} = t - \sqrt{X}$ с предикатом $P(X) = (X \le 0)$ использование метода деления вызовет прерывание. В этих слу-

чаях необходимо использовать специальные алгоритмы приближения к точке переключения «слева» [88].

При использовании стандартных численных методов всегда имеется опасность пропуска точки переключения для немонотонных функций (Рис. 66).

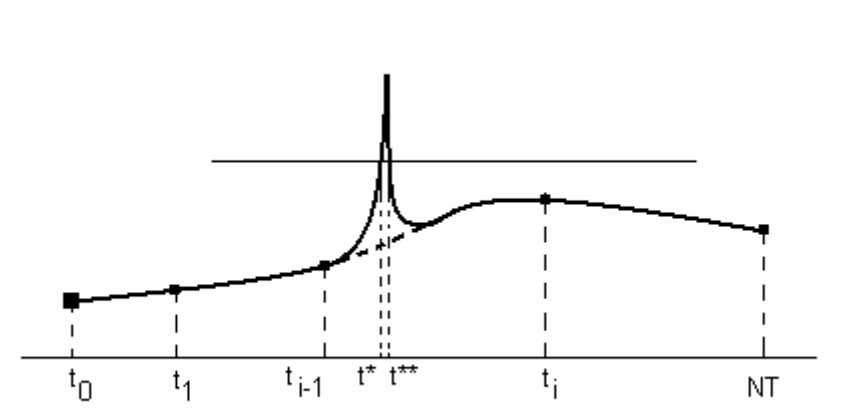


Рис. 66

Эта проблема особенно актуальна для «скрытой» модели, поскольку в визуальной модели велика вероятность того, что величина шага численного метода будет ограничена шагами визуализации и привязки к реальному времени.

Следует также отметить, что некоторые проблемы могут возникать при поиске точек переключения в так называемых «мультиагентных» моделях, в которых независимо и параллельно функционирует очень большое число (до миллионов) взаимодействующих объектов. В такой модели поиск точки переключения методом деления может приводить к огромным вычислительным затратам. В то же время семантика этих моделей (обычно это модели больших организационных систем), как правило, такова, что ближайшая точка переключения зависит или существенно зависит только от решения лишь немногих уравнений. Поэтому для гибридных «мультиагентных» моделей целесообразно проводить динамический анализ структуры системы уравнений, осуществлять эффективный поиск точки переключения и затем однократно находить решение всей совокупной системы уравнений в этой точке.

Еще одной проблемой определения точек переключение является оценка влияния точности решения и точности задания параметров модели на качественный характер фазовой траектории.

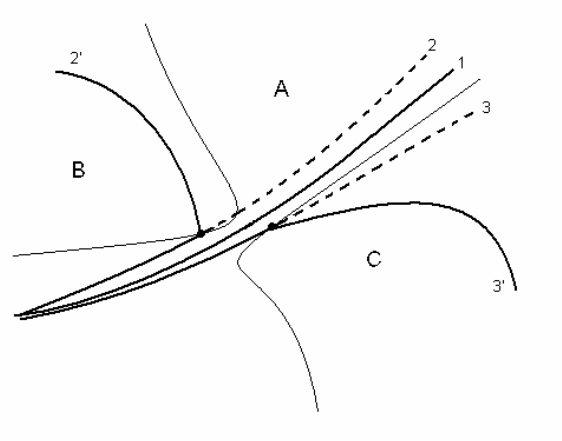


Рис. 67

На Рис. 67 показан случай, когда небольшие отклонения (траектории 2 и 3) от идеальной траектории 1 могут приводить к качественно различному поведению модели (траектории 2’ и 3’). Попытки применить классические подходы к анализу устойчивости гибридных систем [74] пока дают весьма ограниченные результаты. Более перспективным представляется применение метода символического анализа [112].

Реализация условных уравнений.

При использовании условных уравнений вида

```
if <условие> then <выражение1> else < выражение2>
```

в ориентированных компонентах совокупность искомых переменных не изменяется. Однако, даже в этом случае переключение логических ветвей условного выражения в процессе решения совокупной системы уравнений совершенно недопустимо, так как возникает разрыв значений переменных или значений их производных. Следовательно, такое переключение должно восприниматься как дискретное событие, после которого численный метод перезапускается с новыми начальными значениями переменных и их производных. Для этого при генерации кода модели для каждого условного уравнения формируется специальный логический предикат, который становится истин-

ным, если по сравнению с предыдущим вычислением произошло переключение ветвей.

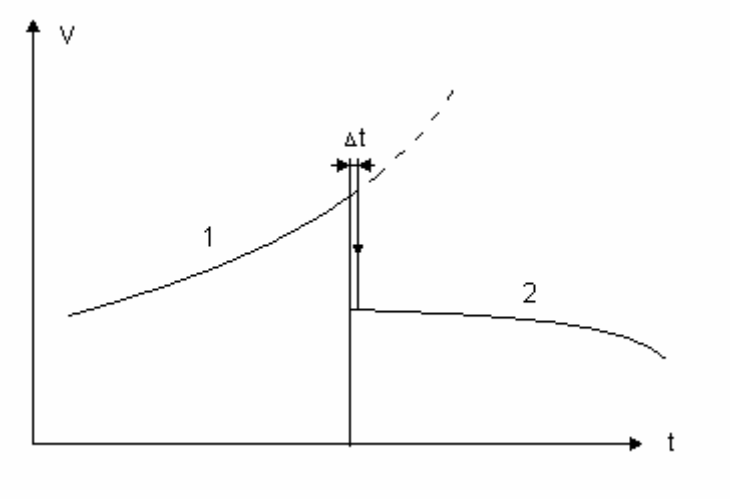


Рис. 68

Объединение по «или» этих предикатов для всех условных уравнений в текущей совокупной системе уравнений используется СМЕ как дополнительный признак наличия точки переключения. Точка переключения ищется с помощью внешнего алгоритма, описанного выше. При этом во время поиска используется только первая ветвь условного выражения (Рис. 68), так что при поиске точки переключения никаких разрывов не возникает.

Реализация функции временной задержки в гибридной модели.

Особым видом функциональной зависимости, поддерживаемой большинством пакетов моделирования, является т.н. «чистая временная задержка». В языке MVL имеется предопределенная функция

$delay(X,\tau)$

где где X - переменная типа double или vector, а τ - выражение типа double. Функция возвращает соответственно результат типа double или vector, равный значению переменной X в момент $t-\tau$, где t - текущее модельное время. На интервале $0 \le t \le \tau$ функция возвращает начальное значение переменной X (если это локальная переменная поведения, то это справедливо для

локального времени поведения). В общем случае с переменной X может быть связано m функций задержки вида $delay(X,\tau_i) \mid i \in 1..m$.

Если значение переменной X вычисляется численно, то реализация функции задержки для непрерывных моделей сводится к корректной интерполяции, выполняемой обычно внутри специальных численных методов [65,66]. Однако, для гибридных моделей такое решение невозможно, поскольку значение переменной X может претерпевать разрывы во временных щелях, то есть является многозначной в непрерывном времени. Результат функции задержки должен воспроизводить такую же последовательность значений в гибридной времени со сдвигом τ в непрерывном времени (Рис. 69).

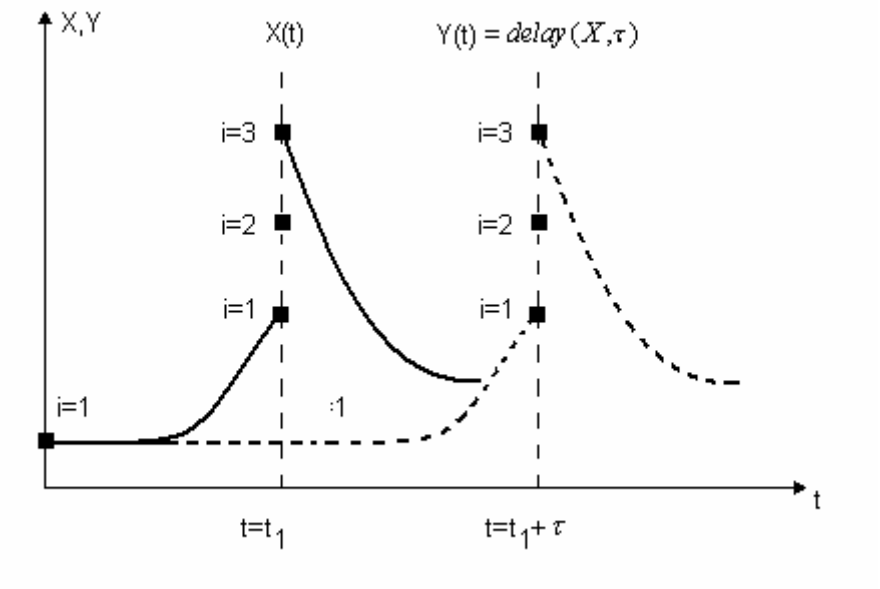


Рис. 69

Таким образом, в гибридной модели функция временной задержки приводит также к задержанным дискретным событиям и поэтому может быть использован только внешний по отношению к численному методу алгоритм ее реализации.

Хранение значений переменной.

Последовательность замеров значений, присваиваемых переменной X, хранится непосредственно в программном объекте $\hat{X}$, соответствующем этой переменной. Каждый замер хранит время присвоения этого значения, само значение и два специальных признака – признак временного замера и признак временной щели. Добавление замеров начинается после первого обращения к любой функции задержки, связанной с данной переменной. При этом в список замеров сразу заносится текущее значение переменной и текущее время. Добавление очередного замера производится при любом присваивании переменной X. Если этот замер первый во временной щели, то в списке КАЛЕНДАРЬ формируется m отложенных на τ_i элементов типа «отложенное дискретное событие».

Новый замер добавляется в список замеров и производится удаление ненужных замеров. Замер полагается ненужным в следующих случаях:

- замер временный внутри шага численного метода и шаг завершен;
- замер соответствует времени, большему чем текущее (численный метод закончил пробные вычисления и вернулся назад);
- замер сделан раньше, чем $t - 2 \cdot \max_{i=1..m} \tau_i$.

Вычисление задержанного значения.

Если в объекте $\hat{X}$ установлен признак задержанного дискретного события DE, то значением функции является специальное значение G_X.

Если $t - \tau < tF + \tau$, т.е. мы попадаем на участок «разгона», то значением функции является vF, где vF и tF соответственно значение и время для первого замера.

В противном случае определяется нужный участок списка замеров (на нем не должно быть временных щелей) и производится интерполяция для значения аргумента $t - \tau$. Интерполяция производится по возможности по 10 точ-

кам (5 слева и 5 справа от найденного участка, если это позволяет отсутствие временных щелей).

Обработка задержанного дискретного события.

Когда DME извлекает из списка КАЛЕНДАРЬ специальный элемент типа «отложенное дискретное событие для переменной X»., он вызывает соответствующий метод объекта $\hat{X}$

В списке замеров находится серия замеров, относящихся к нужной временной щели в момент $t-\tau$. Далее производятся следующие действия:

```
DE:=true;
for i in серия_замеров loop
  замер:= серия_замеров[i];
   G_X :=замер.значение;
  -- обращения к DME
   найтиСогласованныеЗначенияПеременных;
  выполнитьВсеМгновеннныеДействия;
end loop;
DE:=false;
```

Все обращения к функции задержки во время выполнения мгновенных действий будут возвращать значение DE.

Процессы обновления диаграмм.

Поскольку поддержка активного вычислительного эксперимента требует отображения информации в ходе прогона модели, а не после него, в визуальной модели для каждого окна временной или фазовой диаграммы создается экземпляр «служебного» процесса, карта состояний которого приведена на Рис. 70.

Таким образом, для отображения в окне диаграммы согласованных значений набора переменных $V=\{v_i \mid i=1..n\}$, сопоставленного этому окну, с шагом отображения h генерируется «служебное» дискретное событие. Выбор шада отображения h должен определяться масштабом диаграммы и характером изменения отображаемых переменных.

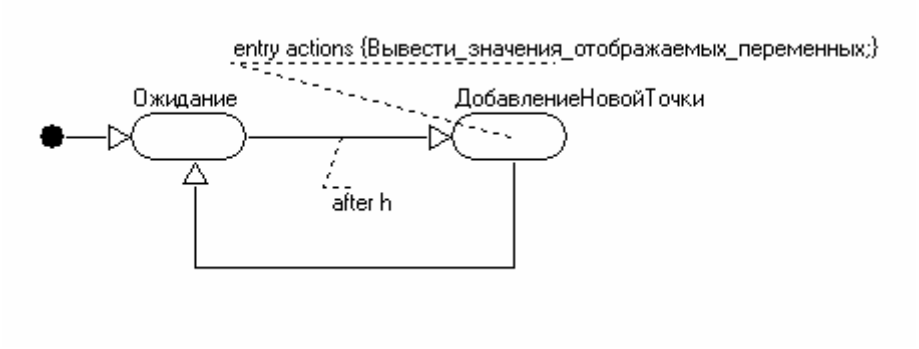
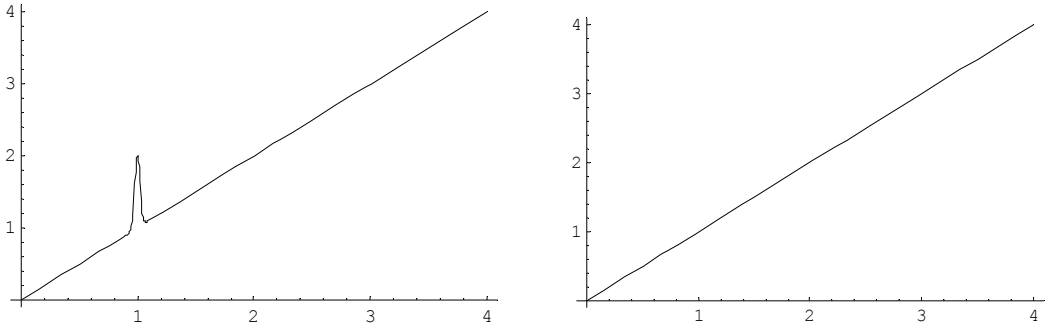


Рис. 70

В идеальном случае пакет моделирования исследовательского типа должен обеспечивать автоматическое отображение на диаграммах всех качественных особенностей фазовой траектории модели (например. очень коротких выбросов или высокочастотных составляющих) [36]. На практике выполнить это требование чрезвычайно сложно (например, для выбора шага отображения h может потребоваться решать задачу определения глобального экстремума функции на интервале). В настоящее время задача автоматического определения точек отображения для модели исследовательского типа не решена (следует отметить, что эта проблема имеет место как для активного, так и для пассивного вычислительного эксперимента). Поэтому существует определенный риск иногда получить даже в самых известных пакетах моделирования вместо правильной зависимости на Рис. 71а неверную зависимость на Рис. 71б. Причем это никак не будет связано с качеством используемых численных методов: при надлежащем выборе точек отображения все особенности обычно легко выявляются.

Поэтому в пакете MVS для временных диаграмм принимается $h = \Delta$, где Δ - величина модельного времени, соответствующая одному пикселю диаграммы по оси абсцисс. Таким образом, проблема адекватного отображения перекладывается на пользователя, который должен выбирать правильный масштаб диаграммы по результатам какого-то предварительного анализа. Исполняющая система ограничивается лишь поддержкой автомасштабирования при выходе очередного значения за границы окна.



а) б)

Рис. 71

Для фазовой диаграммы алгоритм определения шага отображения несколько сложнее. Из соображений эффективности следующая точка должна отстоять от предыдущей не менее чем на один пиксель, а из соображений гладкости кривой не более чем на M пикселей (по умолчанию $M = 3$) по любой из осей. В случае отображения на диаграмме нескольких переменных это качается максимального случая. Однако, расстояния на фазовой диаграмме непосредственно не связаны с модельным временем. Поэтому для определения шага отображения необходимо учитывать динамику изменения переменных, изображаемых на фазовой диаграмме. Для этого в каждой точке отображения t_j для каждой отображаемой переменной v_i вычисляются оценки величины первой и второй производной этой переменной по времени

$$d1_i(t_j) = \frac{v_i(t_j) - v_i(t_{j-1})}{h}$$

$$d2_i(t_j) = \frac{d1_i(t_j) - d1_i(t_{j-1})}{h}$$

Далее предполагается, что вторая производная является постоянной и решаются квадратные уравнения

$$\pm L_i = d1_i(t_j) \cdot h_i + \frac{d2_i(t_j) \cdot h_i^2}{2}$$

где L_i - значение переменной v_i, соответствующее M пикселям в соответствии с выбранным масштабом, относительно величины h_i (используется наи-

меньший положительный корень). В качестве следующего значения шага отображения используется величина $h = \min_{i=1..n} h_i$. В начальный момент значение шага отображения полагается равным некоторой достаточно малой константе h_0. Конечно, эта оценка может оказаться завышенной при резком изменении величины второй производной, в этом случае необходимо определить промежуточные точки с помощью интерполяции. Оценка шага должна начинаться заново в случае дискретного скачка значения отображаемой переменной во временной щели. Опыт эксплуатации пакета MVS показывает, что такое несложное решение дает вполне приемлемые результаты при построении фазовых диаграмм в ходе прогона модели.

Процесс синхронизации с реальным временем.

Необходимость этого служебного процесса в визуальной модели (в «скрытой» модели этот процесс отсутствует) обусловлена двумя причинами. Первая заключается в том, что для некоторых окон (к ним относятся окна 2D и 3D-анимации, а также окна переменных и сам индикатор модельного времени) желательно выполнять обновление с некоторой фиксированной частотой в реальном времени. В противном случае при длительных вычислениях на некоторых участках фазовой траектории у пользователя может возникнуть впечатление, что модель «зависла».

Второй причиной является необходимость обеспечивать продвижение модельного времени в визуальной модели даже в те моменты, когда в модели нет ни одного активного объекта (это вполне нормальная ситуация для моделей систем с переменным составом). В «скрытой» модели эта проблема просто переносится на использующее ее приложение. «Классические» пакеты непрерывно-дискретного моделирования [50,16] требуют добавления в такие модели специальной компоненты, моделирующей «окружающую природу». Концепция же активного вычислительного эксперимента предполагает, что эта компонента («виртуальный стенд», «верстак», «площадка для игр»)

должна быть изначально встроена в исполняющую систему пакета моделирования.

Карта состояний для этого процесса соответствует Рис. 70. Единственным отличием от процессов обновления диаграмм является то, что в момент дискретного события «синхронизация с реальным временем» обновляются разом все окна реального времени.

Возможны два режима синхронизации с реальным временем и соответственно два алгоритма определения шага синхронизации h:

1) с постоянным соотношением K между реальным и модельным временем (по умолчанию $K=1$, то есть моделирование в реальном масштабе времени);

2) режим «так быстро как возможно».

По умолчанию визуальная модель стартует во втором режиме синхронизации.

В первом режиме между реальным и модельным временем должна существовать линейная зависимость $t_i = t_i^R \cdot K$, где t_i и t_i^R - соответственно модельное и реальное время события E_i данного процесса. Предполагается, что в реальном времени события синхронизации должны появляться с некоторой частотой F Гц (по умолчанию F=10). Поэтому в этом режиме $h = \frac{K}{F}$. В конкретном прогоне модели в зависимости от производительности компьютера, на котором осуществляется моделирование, событие E_i наступает в общем случае в некоторый момент реального времени t_i^*, в общем случае отличный от t_i^R. Если $t_i^R \geq t_i^*$, то это означает, что моделирование на интервале между событиями E_{i-1} и E_i осуществлялось быстрее реального времени. В этом случае функциональные действия события E_i будут предваряться приостановкой процесса моделирования на $t_i^R - t_i^*$ в реальном времени. Если $t_i^R < t_i^*$, то это означает, что моделирование на интервале между событиями E_{i-1} и E_i осуществлялось медленнее реального времени. В этом случае процесс синхрониза-

ции пытается снизить частоту F. Если частота уменьшается до 3 Гц, а ситуация не изменяется, то это означает, что заданное соотношение K не может быть выдержано на данном компьютере. Пользователь информируется об этом с помощью изменения цвета на индикаторе модельного времени. Визуальная модель автоматически переводится в первый режим синхронизации при появлении окна анимации, поскольку для правильного восприятия динамических образов необходимо линейное соотношение между модельным и реальным временем.

Во втором режиме синхронизации величина K изменяется при планировании следующего события. Осуществляется автоподстройка под текущее сочетание требуемых вычислительных затрат (эти затраты могут существенно изменяться в различные моменты вычислительного эксперимента) и имеющейся производительности компьютера (эта величина также может изменяться в зависимости от других приложений, одновременно выполняемых операционной системой). При этом целью регулирования является стабилизация периода следования события синхронизации в реальном времени относительно величины $\tau = \frac{1}{F}$ при отсутствии принудительного простоя компьютера. При планировании события E_{i+1} сначала замеряется фактическое время Δ_i, затраченное на продвижение модели от E_{i-1} к E_i при значении коэффициента K равном K_i ($K_0 = 1$). Затем вычисляется новое значение $K_{i+1} = K_i \cdot \frac{\tau}{\Delta_i}$. Таким образом, коэффициент K автоматически увеличивается, если при старте модели удается осуществлять моделирование быстрее реального времени, и уменьшается в противном случае.

Процесс останова по условию.

В процессе отладки визуальной модели бывает чрезвычайно удобно задавать некоторую совокупность условий $\{D_i \mid i = 1..m\}$, при выполнении любого из которых происходит останов. Условиями могут быть как логический предикат произвольного вида над переменными модели, так и специальные

условия – срабатывание определенного перехода или попадание в определенное состояние. На Рис. 72 показана карта состояния процесса останова по условию, где $D = \bigcup_{i=1..m} D_i$.

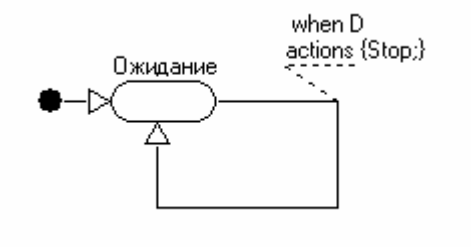


Рис. 72

Интерактивное взаимодействие с пользователем.

В ходе активного вычислительного эксперимента пользователь может в любой момент воздействовать на визуальную модель (изменить значение переменной, послать сигнал и т.п. напрямую или через интерактивные анимационные элементы). Например, пользователь может подбирать значения параметров регулятора следящей системы, просто вращая соответствующие «ручки» на виртуальном пульте управления и наблюдая на графике реакцию управляемого объекта на тестовое воздействие. Для исполняющей системы сигналом о вмешательстве пользователя является появление соответствующего сообщение от оконного элемента. Это сообщение может появиться в совершенно произвольный момент времени, во временной щели или на интервале продвижения непрерывной составляющей модельного времени. Для ядра исполняющей системы это сообщение является некоторым внешним событием, которому сопоставлена некоторая последовательность мгновенных действий. В общем случае это событие не может быть обработано немедленно: предварительно должно быть вычислено текущее согласованное значение переменных модели и осуществлен переход к обработке временной щели. Поэтому при возникновении внешнего события ссылка на соответствующую последовательность мгновенных действий заносится в СПИСОК_ВНЕШНИХ_СОБЫТИЙ (операции с этим списком защищены от параллельного выполнения) и устанавливается признак появления внешнего

события. В общем случае до начала обработки может появиться несколько внешних событий (например, в результате перемещения с помощью мыши ползунка линейного регулятора).

Если в момент поступления внешнего события модель находилась на интервале продвижения непрерывного времени, то по окончании очередного шага интегрирования процедура анализа результатов шага замечает установленный признак внешнего события и прекращает процесс решения. CME завершает работу с признаком «дискретное событие».

DME начинает обработку временной щели, проверяя в начале каждого цикла продвижения дискретной составляющей модельного времени наличие установленного признака внешнего события. Если признак установлен, то DME последовательно выбирает элементы из СПИСКА_ВНЕШНИХ_СОБЫТИЙ, выполняет соответствующую последовательность действий и затем обрабатывает возникающую цепочку дискретных событий.

Распределенные модели гибридных систем.

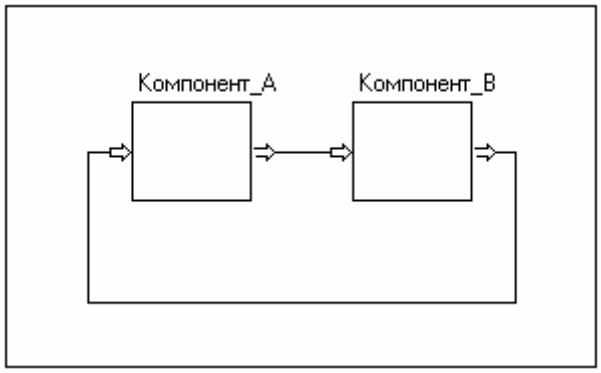
Под распределенным моделированием понимают выполнение отдельных составляющих компьютерной модели на различных компьютерах, на различных процессорах или на одном и том же процессоре в режиме разделения времени [101]. Распределенное моделирование обычно используется для:

1) повышения суммарного быстродействия компьютерной модели;
2) объединения независимых компьютерных моделей, созданных различными разработчиками с использованием различных инструментальных средств.

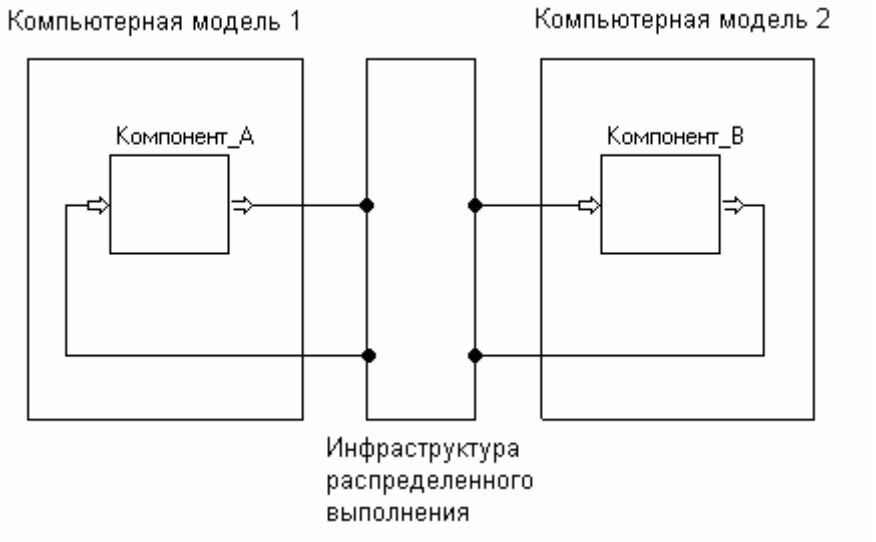
Для разработки сложной системы управления вторая цель является более значимой, поскольку разработать достаточно точные модели для подсистем различной физической природы в рамках одного проекта практически невозможно [101]. Следует отметить, что при наличии средств автоматического

синтеза выполняемых моделей вполне возможна унификация моделей подсистем на уровне математических описаний (при наличии, конечно, желания разработчиков открывать эти описания). Интересное решение проблемы объединения разнородных моделей предлагается в системе PTOLEMY II [115].

В распределенной модели неизбежно появляется некоторый блок-посредник, называемый обычно «инфраструктура распределенного выполнения» (Рис. 73).



а)



б)

Рис. 73

Одним из наиболее распространенных и перспективных в настоящее время является интерфейс распределенного выполнения HLA («High Level Architecture») [101,96]. Однако, этот стандарт (как и другие известные стандарты распределенного моделирования ALSP и DMS) предназначен только для сис-

тем с дискретным поведением. В ходе разработки пакета AnyLogic была предпринята попытка расширения стандарта HLA применительно к моделированию гибридных систем [79]. Рассматривались варианты «разрезания» компьютерных моделей гибридных систем по дискретным и по непрерывным направленным связям. В случае «разрезания» только дискретных связей изменения касаются лишь механизма продвижения общего модельного времени: в каждой компьютерной модели сначала проводится «пробный» просчет фазовой траектории до ближайшего запланированного в КАЛЕНДАРЕ события, чтобы проверить, не возникнет ли на этом интервале другое дискретное событие, связанное с ожидающим предикатом. Для случая «разрезанных» непрерывных связей использованы специальные агенты, выполняющие линейную и квадратичную аппроксимацию значений передаваемой по этой связи переменной.

Комплексный моделирующий стенд.

Комплексный моделирующий стенд (КМС) включает в себя компьютерные модели и некоторые реальные устройства разрабатываемой системы управления. Такие модели часто называют также HIL-моделями («Hardware In the Loop») [101] или полунатурными. Поскольку в данной работе любая независимо функционирующая компьютерная модель трактуется как физическое устройство - имитатор, то на взгляд автора комплексный моделирующий стенд следует считать частным случаем распределенной модели, в которой некоторые элементы являются реальными аппаратными блоками системы управления.

В процессе проектирования наиболее часто используются две схемы комплексного моделирующего стенда:

1) устройства управления и некоторые объекты управления моделируются, а некоторые объекты управления представлены физическими устройствами (Рис. 74);

2) некоторые устройства управления представлены реальными встроенными ЭВМ с соответствующим программным обеспечением, а остальная часть системы управления моделируется (Рис. 75).

. В обеих этих схемах «разрезаются» только дискретные связи.

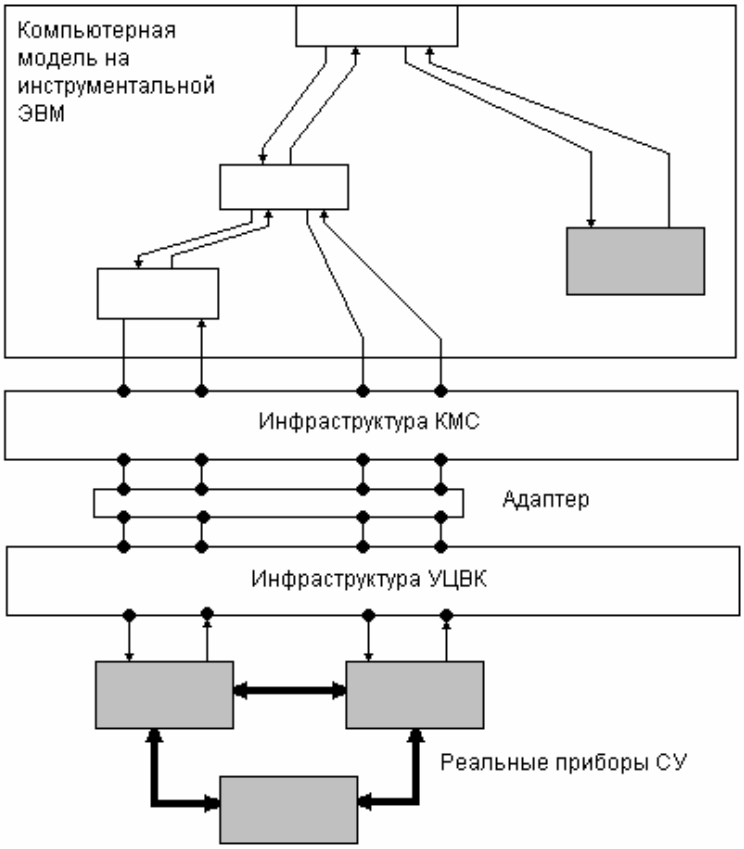


Рис. 74

Комплексный моделирующий стенд как распределенная модель всегда имеет как минимум два компонента; компьютерную модель на инструментальной ЭВМ и адаптер для реального УЦВК, поддерживающий интерфейс распределенного выполнения. Инфраструктура комплексного моделирующего стенда помимо функций распределенного выполнения должна обеспечивать привязку к реальному или кусочно-реальному (программное обеспечение встроенной ЭВМ может выполняться в режиме отладки) времени [101]. Таким образом, моделирование гибридной системы в составе комплексного моделирующего стенда можно считать специальным случаем распределенного моделирования гибридной системы, рассмотренного выше.

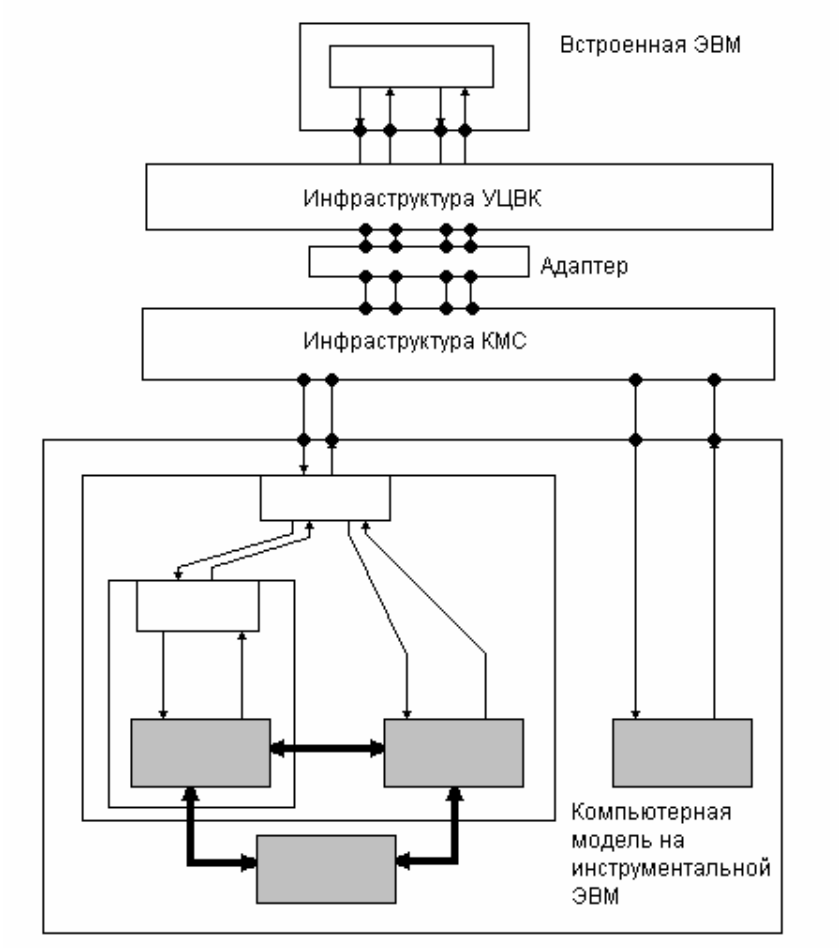


Рис. 75

Язык программирования сверхвысокого уровня.

Метод автоматического синтеза выполняемых моделей гибридных систем, предложенный в данной работе, позволяет говорить о возможности создания новой «сквозной» технологии проектирования сложных систем управления. Основная идея этой технологии заключается в следующем. Объектно-ориентированную математическую модель системы управления, создаваемую на этапе анализа, можно рассматривать как описание (спецификацию) проектируемой системы на некотором формальном языке сверхвысокого уровня. Любая компьютерная модель, однозначно соответствующая этой математической модели, является ее физической реализацией. На этапе анализа эта компьютерная модель строится целиком на базе инструментальной ЭВМ

и ее системного программного обеспечения (например, на платформе Intel-Windows).

На последующих этапах разработки могут использоваться различные промежуточные распределенные компьютерные модели в составе КМС. На Рис. 76 показана распределенная модель системы управления, в которой все устройства управления представлены компьютерными моделями, а объекты управления – реальными штатными объектами.

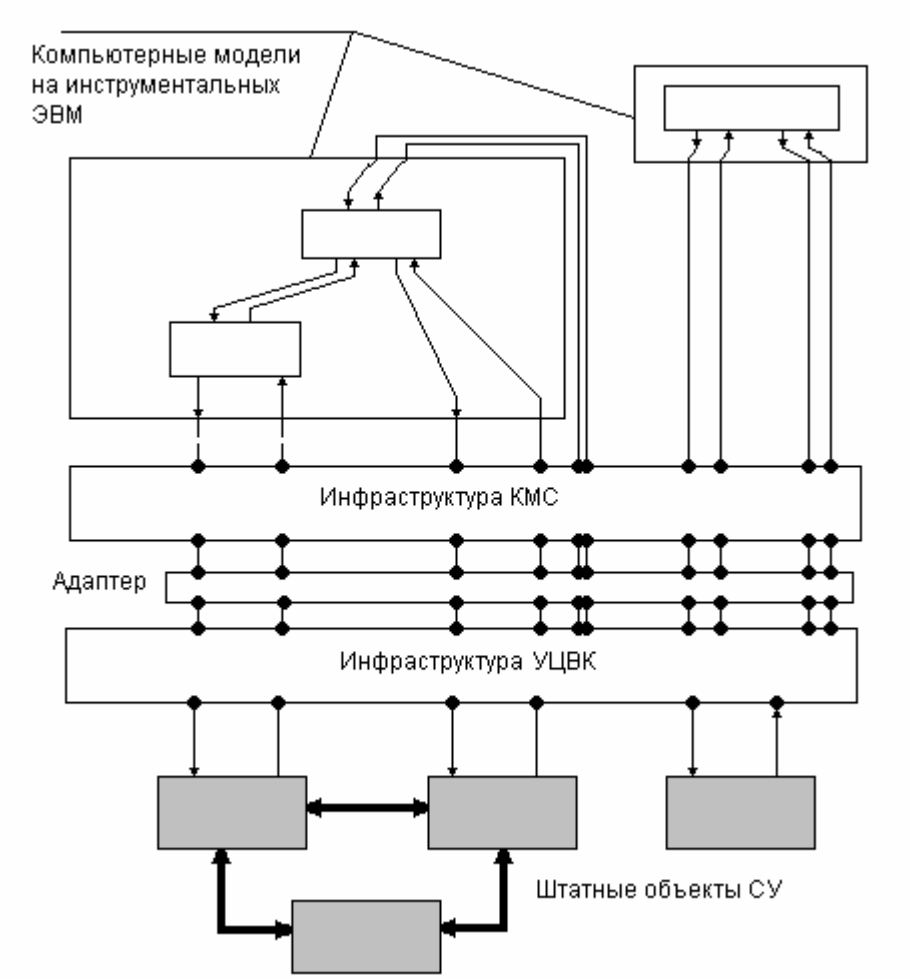


Рис. 76

Предполагается, что распределение устройств управления по отдельным компьютерным моделям соответствует их штатному распределению по встроенным ЭВМ. Кроме того, в этой модели инфраструктура КМС не использует свою локальную сеть и все обмены данными идут через штатную магистраль УЦВК. Эта система является физической и при достаточной производительности инструментальной ЭВМ может имитировать систему

управления (она не может являться штатной только из-за несоответствия инструментальных ЭВМ условиям применения СУ).

- На Рис. 77 показана реальная система, в которой устройства управления реализованы программно на встроенных ЭВМ.

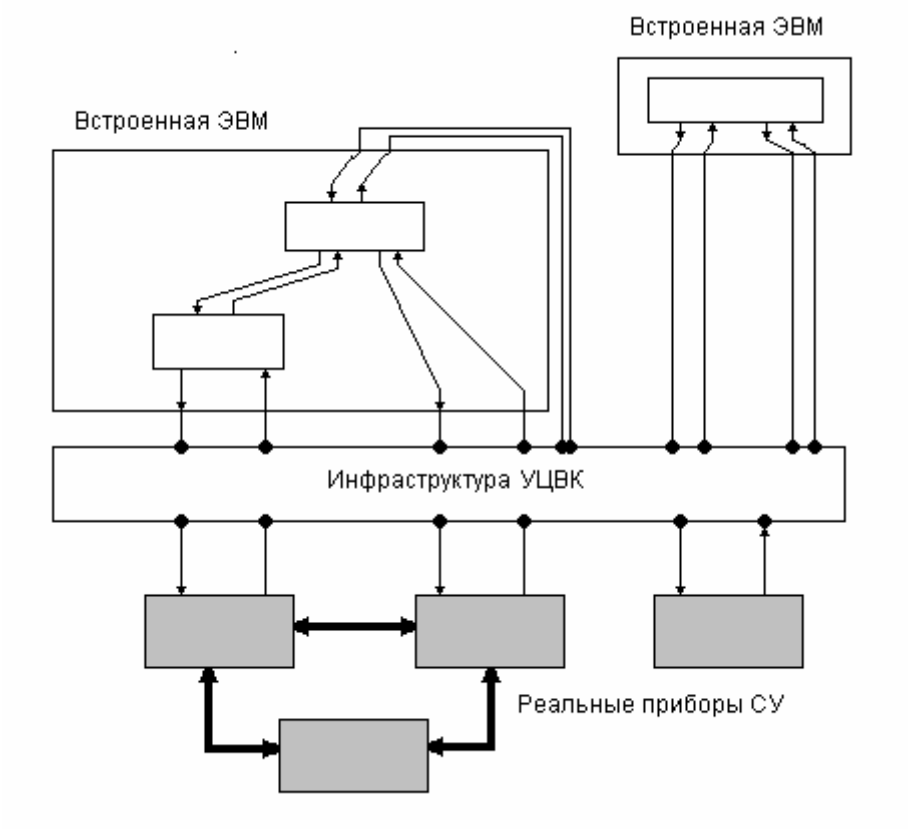


Рис. 77

Легко видеть, что рассмотренная выше модель системы отличается от реальной системы только использованием инструментальной ЭВМ и инфраструктуры КМС. Предположим, что:

- все функции управления реализуются программно на встроенных ЭВМ;
- эти встроенные ЭВМ достаточно производительны;
- существуют некоторые кросс-компиляторы, поддерживающие для этих ЭВМ какой-либо из объектно-ориентированных языков программирования;
- пакет моделирования включает генератор «кода» выполняемой модели для этого языка;

- создан вариант исполняющей системы пакета моделирования для этого языка и используемой операционной системы реального времени;
- интерфейсы инфраструктур КМС и УЦВК совпадают.

В этом случае ручную разработку прикладного программного обеспечения для встроенных ЭВМ можно просто исключить и использовать вместо него выполняемые модели устройств управления, генерируемые автоматически по их математическим моделям. Язык системно-аналитического моделирования будет выступать в роли языка программирования сверхвысокого уровня для встроенных ЭВМ. Например, для программирования регулятора достаточно будет просто задать соответствующую передаточную функцию или систему нелинейных уравнений, а численная реализация будет построена автоматически. Программирование устройства логического управления сведется к рисованию соответствующей карты поведений. Собственно программирование для встроенной ЭВМ будет включать в себя разработку операционной системы реального времени (если не используется стандартная), разработку средств поддержки инфраструктуры УЦВК для штатной магистрали обмена и разработку специальной исполняющей системы для пакета моделирования. Попытка создания такой многоуровневой структуры программ для одного частного случая была предпринята в работе [40].

Естественно, автоматически сгенерированное объектно-ориентированное прикладное программное обеспечение в среднем будем работать существенно медленнее, чем созданное вручную с использованием процедурных языков программирования. Однако, на протяжении последних 15-ти лет производительность ЭВМ увеличивалась примерно в 100 раз за каждые 5 лет [42] и уже достигла или достигнет в недалеком будущем уровня, когда этой разницей можно будет пренебречь. Лозунгом любой «сквозной» технологии может служить подзаголовок книги [101]: «Build Better Embedded Systems Faster». Использование для разработки прикладного программного обеспечения встроенных ЭВМ языка сверхвысокого уровня позволит прин-

ципиально сократить сроки разработки системы управления и повысить ее надежность.

Заключение.

В работе изложены теоретические и методические основы и инженерные методики построения инструментальных средств автоматизации объектно-ориентированного моделирования сложных динамических систем на основе формализма гибридного автомата.

Основные теоретические результаты реализованы в семействе пакетов автоматизации моделирования Model Vision: Model Vision for DOS [24], Model Vision for Windows [25] и Model Vision Studium [26]. В настоящее время готовится к выпуску пакет Model Vision 4. Пакет Model Vision Studium является некоммерческим программным обеспечением и свободно распространяется через Интернет (русскоязычная версия на образовательном сайте www.exponenta.ru в разделе «Другие пакеты»). Кроме того, различные версии пакета выпускались в качестве приложений на компакт-диске вместе с книгами [4,34]. Универсальность входного языка и широкие анимационные возможности делают семейство Model Vision удобным для проведения вычислительных экспериментов в рамках исследований в различных областях техники, проводимых в ВУЗ'ах [13,14,38,39,99]. Простота пользовательского интерфейса, поддержка активного вычислительного эксперимента, а также невысокие требования к компьютеру делают семейство Model Vision привлекательным для использования в учебном процессе. [14,48] Пакет Model Vision Studium используется в настоящее время примерно в двух десятках ВУЗ'ов. Пакет также использовался рядом предприятий в ходе НИР и ОКР.

Дальнейшие исследования должны, на взгляд автора, проводиться по двум основным направлениям:

1) качественный анализ сложных динамических систем;
2) разработка «сквозной» технологии анализа и синтеза сложной технической системы на основе ее объектно-ориентированной модели.

Литература.

1. Андронов А.А., Леонтович Е.А., Гордон М.И., Майер А.Г. Качественная теория динамических систем 2-го порядка. М.: Наука, 1966. -.568 с.
2. Арайс Е.А., Дмитриев В.М.. Автоматизация моделирования многосвязных механических систем. М.: Машиностроение, 1987.- 240с.
3. Баяндин Д.В., Кубышкин А.В., Мухин О.И., Рябуха А.А. Математическое моделирование в системе "Stratum Computer" // Труды Всероссийской научно-практической конференции "Проблемы образования, научно-технического развития и экономики Уральского региона". Березники, 1996, с.80-81.
4. Бенькович Е.С., Колесов Ю.Б., Сениченков Ю.Б. Практическое моделирование сложных динамических систем. С. Петербург, БХВ, 2001.- 441с.
5. Безкоровайный М.М., Костогрызов А.И., Львов В.М. Инструментально-моделирующий комплекс для оценки качества функционирования информационных систем «КОК». Руководство системного аналитика. – М.: Синтег, 2000. – 116с.
6. Бизли Д.. Язык программирования PYTHON, Киев, ДиаСофт, 2000. – 336 с.
7. Боггс У, Боггс М. UML и Rational Rose, М.: Лори, 2000. - 582с.
8. Борщев А.В., Карпов Ю.Г., Колесов Ю.Б. Спецификация и верификация систем логического управления реального времени. – Системная информатика, вып.2, Системы программирования. Теория и приложения. – Новосибирск: ВО «Наука», 1993, с. 113-147.
9. Бромберг П.В. Матричные методы в теории релейного и импульсного регулирования. М.: Наука, 1967. - 323 с.
10. Бусленко Н.П.. Моделирование сложных систем. М.:Наука,1978.-384 с.
11. Буч Г. Объектно-ориентированный анализ и проектирование с примерами на C++, 3-е изд. / Пер. с англ. – М.: «Издательство Бином», СПб.: «Невский диалект», 2001 – 560с.
12. Буч Г., Рамбо Д., Джекобсон А. Язык UML. Руководство пользователя: Пер. с англ. – М.: ДМК, 2000. – 432с.
13. Васильев А.Е., Киричков А.В., Леонтьев А.Г. Исследование мехатронных систем с нечетким управлением с применением пакета Model Vision 3.0. // Гибридные системы. Model Vision Studium: Труды междунар. науч.-технич. конф. СПб.: Изд-во СПбГТУ , 2001. – с.46-50.

14. Васильев А.Е., Леонтьев А.Г. Применение пакета Model Vision Studium для исследования мехатронных систем. // Гибридные системы. Model Vision Studium: Труды междунар. науч.-технич. конф. СПб.: Изд-во СПбГТУ , 2001. – с.51-52.
15. Вендров А.М. CASE-технологии: Современные методы и средства проектирования информационных систем. М.: Финансы и статистика, 1998. - 176с.
16. Глушков В.М., Гусев В.В., Марьянович Т.П., Сахнюк М.А. Программные средства моделирования непрерывно-дискретных систем. – Киев: Наукова думка, 1975. – 152с.
17. Гома Х. UML. Проектирование систем реального времени, параллельных и распределенных приложений: Пер. с англ. – М.: ДМК Пресс, 2002. – 704с.
18. Гультяев А.К.. MATLAB 5.3. Имитационное моделирование в среде Windows, М.: Корона принт, 2001. – 400с.
19. Дал У., Мюрхауг Б., Нюгород К. СИМУЛА-67. Универсальный язык программирования. М.: Мир, 1969. - 99с.
20. Дмитриев А.К., Мальцев П.А. Основы теории построения и контроля сложных систем. Л.: Энергоатомиздат, 1988.- 192 с.
21. Дьяконов В. Mathematica 4: учебный курс. – СПб: Питер, 2002. – 656 с
22. Емельянов Е.С.. Системы автоматического управления с переменной структурой. М.: Наука, 1967. - 335 с.
23. Емельянов С.В, Коровин С.К. Новые типы обратной связи. М.: Наука, 1997. 352 с.
24. Инихов Д.Б, Инихова М.А., Колесов Ю.Б., Сениченков Ю.Б. Свидетельство об официальной регистрации программы для ЭВМ «Model Vision ver. 1.5» №930033. – Москва, РосАПО, 14.10.1993.
25. Инихов Д.Б, Инихова М.А., Колесов Ю.Б., Сениченков Ю.Б. Свидетельство об официальной регистрации программы для ЭВМ «Model Vision for Windows» №950277. – Москва, РосАПО, 04.08.1995.
26. Инихов Д.Б,, Колесов Ю.Б., Сениченков Ю.Б. Свидетельство об официальной регистрации программы для ЭВМ «Model Vision Studium версия 3.0» №990643. – Москва, Роспатент, 6.09.1999.
27. Калман Р., Фалб П., Арбиб М.. Очерки по математической теории систем. М.: Мир, 1971. 400 с.
28. Касти Дж. Большие системы. Связность, сложность и катастрофы. = М.: Мир, 1982. – 216с.
29. Киндлер Е. Языки моделирования: Пер. с чеш. – М.: Энергоатомиздат, 1985. – 389с.

30. Козлов О.С., Медведев В.С. Цифровое моделирование следящих приводов. // В кн.: Следящие приводы. В 3-х т. /Под ред. Б.К. Чемоданова. М.: Изд. МГТУ им. Н.Э. Баумана, 1999. Т. 1. С. 711-806.
31. Колесов Ю.Б. Анализ корректности процессов логического управления динамическими объектами // Известия ЛЭТИ. Сб. научн. Трудов / Ленингр. Электротехнич. Ин-т им. В.И.Ульянова (Ленина). – Л.: 1991. – Вып. 436. – с. 65-70.
32. Колесов Ю.Б. Свидетельство об официальной регистрации программы для ЭВМ «MVBase версия 8.0» №2001610183. – Москва, Роспатент, 21.02.2001.
33. Колесов Ю.Б., Сениченков Ю.Б. Библиотека программ для решения ОДУ. Труды ЛПИ, 462. – С.Пб.: 1996, с. 116-122.
34. Колесов Ю.Б., Сениченков Ю.Б.. Визуальное моделирование сложных динамических систем. Изд. «Мир и Семья & Интерлайн», СПб, 2000, 242с.
35. Колесов Ю.Б., Сениченков Ю.Б. Компьютерное моделирование в научных исследованиях и в образовании. "Exponenta Pro. Математика в приложениях", №1, 2003, с. 4-11.
36. Колесов Ю.Б., Сениченков Ю.Б. Программная поддержка активного вычислительного эксперимента В сб. "Научно-технические ведомости СПбГПУ", №1.2004.
37. Колесов Ю.Б., Сениченков Ю.Б.. Синхронизация событий при использовании гибридных автоматов для численного моделирования сложных динамических систем. В сб. "Научно-технические ведомости СПбГПУ", №1.2004.
38. Колесов Ю.Б., Цитович И.Г. Имитационная модель участка трикотажного производства // Известия ВУЗ’ов. Технология легкой промышленности, 1993, №6, с.56-61.
39. Колесов Ю.Б., Цитович И.Г. Оценка эффективности новой кругловязальной машины с помощью имитационной модели // Известия ВУЗ’ов. Технология легкой промышленности, 1994, №4, с. 72-77.
40. Курочкин Е.П., Колесов Ю.Б. Технология программирования сложных систем управления / ВМНУЦ ВТИ ГКВТИ СССР. – М.: 1990. – 112с.
41. Липаев В.В. Надежность программных средств, М.: Синтег, 1998. – 232с.
42. Липаев В.В. Системное проектирование сложных программных средств для информационных систем. - М.: Синтег, 1999. – 224с.
43. Майо Д. C#: Искусство программирования. Энциклопедия программиста: Пер. с англ. СПб.: «ДиаСофтЮП», 2002. – 656 с.

44. Меерович Г.А.. Эффект больших систем., М.: Знание, 1985. – 231с.
45. Мехатроника: Пер. с япон. / Исии Т., Симояма И., Иноуэ Х., и др. – М.: Мир, 1988. – 387с.
46. Мухин О.И. Компьютерная инструментальная среда "Слоистая машина". Пермь, ППИ, 1991. 122 с.
47. Мухин О.И. Универсальная инструментальная среда "Stratum Computer" - программный продукт нового поколения // Проблемы информатизации высшей школы (бюллетень Госкомвуза РФ). М., ГосНИИ СИ, 1995. Вып.2. 10-1 - 10-4.
48. Петров Г.Н. Использование пакета “Model Vision” для создания компьютерных лабораторных работ. // Гибридные системы. Model Vision Studium: Труды междунар. науч.-технич. конф. СПб.: Изд-во СПбГТУ , 2001. – с.53-54.
49. Подчуфаров Ю.Б.. Физико-математическое моделирование систем управления и комплексов / Под ред. А.Г.Шипунова. – М.: Изд-во физико-математической литературы, 2002. – 168с.
50. Прицкер А. Введение в имитационное моделирование и язык СЛАМ II: Пер. с англ. - М.: Мир, 1987. - 646с.
51. Самарский А.А., Михайлов А.П.. Математическое моделирование: Идеи. Методы. Примеры. - М.: Наука. Физматлит, 1997.-320 с.
52. Семененко М. Введение в математическое моделирование - М.:Солон-Р, 2002. - 112с.
53. Солодовников В.В. Теория автоматического регулирования. М.: Машиностроение, 1976, т.1 – 768 с
54. Теория систем с переменной структурой./ Под редакцией С. В. Емельянова. М.: Наука, 1970. - 590 с.
55. Терехов А.Н., Романовский К.Ю, Кознов Дм. В., Долгов П.С., Иванов А.Н. Объектно-ориентированная методология разработки информационных систем и систем реального времени. // Объектно-ориентированное визуальное моделирование / Под ред. Проф. Терехова А.Н. – СПб: Издательство С.-Петербургского университета, 1999. с.4-20.
56. Трудоношин В.А., Пивоварова Н.В.. Математические модели технических объектов - Мн.: Выш. шк.,1988 - 159с.
57. Уткин В.И. Скользящие режимы в задачах оптимизации и управления. М.: Наука, 1981. – 368 с.
58. Филлиппов А.Ф. Дифференциальные уравнения с разрывной правой частью, М.: Наука, 1985,. - 223 с.

59. Хайрер Э., Ваннер Г. Решение обыкновенных дифференциальных уравнений. Жесткие задачи и дифференциально-алгебраические задачи, М., Мир, 1999,- 685с.
60. Хоар Ч. Взаимодействующие последовательные процессы: Пер. с англ.- М.: Мир, 1989. – 264с.
61. Черемных С.В., Семенов И.О., Ручкин В.С. Структурный анализ систем: IDEF-технологии, М.: Финстат, 2001. - 208с.
62. Черных И.В. Simulink: среда создания инженерных приложений. М.: ДИАЛОГ-МИФИ, 2003. – 496с.
63. Шеннон Р. Имитационное моделирование - искусство и наука. М.: Мир, 1978.- 418с.
64. Шорников Ю.В., Жданов Т.С., Ландовский В.В.. Компьютерное моделирование динамических систем // «Компьютерное моделирование 2003». Труды 4-й межд. научно-техн. конференции, С.Петербург, 24-28 июня 2003г., с.373-380
65. Эльсгольц Л.Э. Дифференциальные уравнения с запаздывающим аргументом. М., Наука, 1965. – 394с.
66. Эльсгольц Л.Э., Норкин С.Б. Введение в теорию дифференциальных уравнений с отклоняющимся аргументом. М., Наука, 1971. – 405с.
67. Юдицкий С.А., Покалев С.С.. Логическое управление гибким интегрированным производством // Институт проблем управления. – Препринт. – М., 1989. – 55с.
68. Andersson M. Omola - An Object-Oriented Language for Model Representation, in: 1989 IEEE Control Systems Society Workshop on Computer-Aided Control System Design (CACSD), Tampa, Florida, 1989.
69. Andersson M. OmSim and Omola Tutorial and User's Manual. Version 3.4., Department of Automatic Control, Lund Institute of Technology, 1995, pp.45.
70. ANSI/IEEE Std 754-1985. IEEE Standard for Binary Floating-Point Arithmetic, 1985.
71. AnyLogic User's Manual. http://www.xjtek.com.
72. Ascher Uri M., Petzold Linda R. Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations. SIAM, Philadelphia, 1998.
73. Astrom K.J., Elmqvist H., Mattsson S.E. Evolution of continuous-time modeling and simulation. The 12th European Simulation Multiconference, ESM'98, June 16-19, Manchester, UK.

74. Avrutin V., Schutz M. Remarks to simulation and investigation of hybrid systems, // Гибридные системы. Model Vision Studium: Труды междунар. науч.-технич. конф. СПб.: Изд-во СПбГТУ , 2001. – с.64-66.

75. Baleani M., Ferrari F., Sangiovanni-Vincentelli A.L., and Turchetti C. HW/SW Codesign of an Engine Management System. In Proc. Design Automation and Test in Europe, DATE'00, Paris, France, March 2000, pp.263-270.

76. Baleani M., Gennari F., Jiang Y., Patel Y., Brayton R.K., and Sangiovanni-Vincentelli A.L.. HW/SW Partitioning and Code Generation of Embedded Control Applications on a Reconfigurable Architecture Platform. In Proc. International Symposium on Hardware/Software Codesign, CODES'02, Estes Park, Colorado, May 2002, pp. 151-156.

77. Booch G. Object-Oriented Analysis and Design with Applicatons, 2nd ed. Redwood City, California, Addison-Wesley Publishing Company, 1993.

78. Booch G.., Jacobson I., Rumbaugh J. The Unified Modeling Language for Object-Oriented Development. Documentation Set Version 1.1. September 1997.

79. Borshchev A., Karpov Yu., Kharitonov V. Distributed Simulation of Hybrid Systems with AnyLogic and HLA // Future Generation Computer Systems v.18 (2002), pp.829-839.

80. Borshchev A, Kolesov Yu., Senichenkov Yu. Java engine for UML based hybrid state machines./In Proceedings of Winter Simulation Conference, Orlando, California, USA, 2000. p. 1888-1897.

81. Brenan K.E., Campbell S.L., Petzold L.R. Numerical solution of initial-value problems in differential-algebraic equations. North-Holland, 1989, 195 p.

82. Bruck D., Elmqvist H., Olsson H., Mattsson S.E. Dymola for multi-engineering modeling and simulation. 2nd International Modelica Conference, March 18-19 2002, Proceedings, pp. 55-1 – 55-8.

83. Bunus P., Fritzson P. Methods for Structural Analysis and Debugging of Modelica Models. 2nd International Modelica Conference, 2002, Proceeding, pp. 157-165.

84. Darnell K., Mulpur A.K.. Visual Simulation with Student VisSim, Brooks Cole Publishing, 1996.

85. Deshpande A., Gullu A., Semenzato L. The SHIFT programming language and run-time system for dynamic networks of hybrid automata. http://www.path.berkeley.edu/shift/publications.html

86. Elmqvist, H., F.E. Cellier, M. Otter, Object-Oriented Modeling of Hybrid Systems, Proc. ESS'93, SCS European Simulation Symposium, Delft, The Netherlands, 1993, pp.xxxi-xli.

87. Elmqvist H., Mattsson S.E., Otter M.. Modelica – the new object-oriented modeling language. The 12th European Simulation Multiconference, ESM'98, June 16-19, Manchester, UK.

88. Esposit J.M., Kumar V., Pappas G.I. Accurate event detection for simulating hybrid systems. Hybrid Systems: Computation and Control, 4th International Workshop, HSCC 2001, Rome, Italy, March 28-30, 2001, Proceedings, pp.204-217.

89. Ferreira J.A., Estima de Oliveira J.P. Modelling hybrid systems using statecharts and Modelica. . In Proc. of the 7th IEEE International Conference on Emerging Technologies and Factory Automation, Barcelona, Spain, 18-21 Oct., 1999, p.1063.

90. Fritzson P., Gunnarson J., Jirstrand M. MathModelica – an extensible modeling and simulation environment with integrated graphics and literate programming/ 2nd International Modelica Conference, March 18-19 2002, Proceedings, pp. 41-54.

91. Fritzson P., Viklund L., Herber J., Fritzson D. Industrial application of object-oriented mathematical modeling and computer algebra in mechanical analysis. In Georg Heeg, Boris Magnosson, and Bertrand Meyer, editors, Technology of Object-Oriented Languages and Systems - TOOLS 7, pp. 167-181. Prentice Hall, 1992.

92. Gollu A., Kourjanski M. Object-oriented design of automated highway simulators using SHIFT programming language. http://www.path.berkeley.edu/shift/publications.html

93. Harel D.. Statecharts: a visual formalism for complex systems. In Science of Computer Programming, North-Holland, Vol.8, No.3, 1987, pp. 231-274.

94. Harel D., Gery E. Executable Object Modeling with Statecharts / Computer, July 1997, pp. 31-42.

95. Hyunok Oh, Soonhoi Ha. Hardware-software cosynthesis of multi-mode multi-task embedded systems with real-time constraints. In Proc. International Symposium on Hardware/Software Codesign, CODES'02, Estes Park, Colorado, May 2002, pp. 133-138.

96. IEEE 1516 HLA Standards.

97. Jacobson I., Cristerson M., Jonsson P., Overgaard G. Object-Oriented Software Engineering: A Use Case Driven Approach. Wokingham, England, Addison-Wesley Publishing Company, 1992.

98. Kesten Y., Pnueli A. Timed and hybrid statecharts and their textual representation. Lec. Notes in Comp. Sci. pp. 591-620, Springer-Verlag, 1992.

99. Khartsiev V.E., Shpunt V.K., Levchenko V.F., Kolesov Yu., Senichenkov Yu., Bogotushin Yu. The modeling of synergetic interaction in Theoretical biology. / Tools for mathematical modelling. St. Petersburg, 1999, p.71-73.

100. Kolesov Y., Senichenkov Y. A composition of open hybrid automata. Proceedings of IEEE Region 8 International Conference «Computer as a tool», Ljubljana, Slovenia, Sep.22-24,. 2003, v.2, pp. 327-331.

101. Ledin J. Simulation Engineering. CMP Books, Lawrence, Kansas, 2001.

102. Maler O., Manna Z., and Pnueli A. A formal approach to hybrid systems. In Proccedings of the REX workshop "Real-Time: Theory in Practice", LNCS. Springer Verlag, New York, 1992.

103. Maler O., Manna Z., and Pnueli A. From timed to hybrid systems. In Proccedings of the REX workshop "Real-Time: Theory in Practice", LNCS. Springer Verlag, New York, 1992.

104. Marca D.A, McGowan C.L. SADT: Structured analysis and design techniques New York: McGraw-Hill, 1988.

105. Mattsson S.E., Elmqvist H., Otter M., Olsson H. Initialization of hybrid differential-algebraic equations in Modelica 2.0. 2nd International Modelica Conference, March 18-19 2002, Proceedings, pp. 9-15.

106. Modelica – a unified object-oriented language for physical systems modeling. Tutorial. Version 1.4, December 15, 2000.

107. Modelica - A Unified Object-Oriented Language for Physical Systems Modeling. Language Specification. Version 2.0, July 10, 2002.

108. Modelica - A Unified Object-Oriented Language for Physical Systems Modeling. Tutorial. Version 2.0, July 10, 2002.

109. Moraleda A.Urquia, Bencomo S. Dormido. Object oriented description of hybrid dynamic systems of variable structure. Departamento de Informatica y Automatica, Facultad de Ciencias, U.N.E.D., Avenida Senda del Rey s/n, 28040 Madrid. Spain, http://www.dia.uned.es

110. Mosterman P.J. An overview of hybrid simulation phenomena and their support by simulation packages. In Hybrid Systems: Computation and Control '99, vol. 1569 in Lecture Notes in Computer Science, Frits W. Vaandrager and Jan H. van Schuppen (eds.), pp. 165-177, 1999.

111. Mosterman P.J. Hybrid dynamic systems: a hybrid bond graph modeling paradigm and its application in diagnosis. Dissertation for the degree PhD of Electrical Engineering/ - Vanderbilt University, Nashvill, Tenneessee, 1997.

112. Osipenko G. Spectrum of a dynamical system and applied symbolic dynamics, Journal of Mathematical Analysis and Applications, v. 252, no. 2, 2000, pp.587-616 .

113. Otter M., Elmqvist H., Mattsson S.E. Hybrid modeling in Modelica based on the synchronous data flow principle. In Proceeding of the 1999

IEEE Symposium on Computer-Aided Control System Design, CACSD'99, Hawai,USA, August 1999.

114. Pantelides C.C. The consistent initialization of differential-algebraic systems. SIAM J. Sci. Stat. Comput. 9(2), 1988, p.213-231.

115. PTOLEMY II HETEROGENEOUS CONCURRENT MODELING AND DESIGN IN JAVA. Edited by: Christopher Hylands, Edward A. Lee, Jie Liu, Xiaojun, Liu, Steve Neuendorffer, Yuhong Xiong, Haiyang Zheng. Department of Electrical Engineering and Computer Sciences University of California at Berkeley, http://ptolemy.eecs.berkeley.edu.. Document Version 2.0.1 for use with Ptolemy II 2.0.1, August 5, 2002.

116. Selic B., Gullekson G., Ward P.T. Real-Time Object-Oriented Modeling. John Wiley & Sons. Inc. 1994.

117. Viklund L., Fritzson P. An object-oriented language for symbolic computation - applied to machine element analysis. In Paul S. Wang, editor, Proceedings of the International Symposium on Symbolic and Algebraic Computation, pp. 397-405. ACM Press, 1992.

118. Yourdon E. Modern structured analysis. Prentice-Hall, New Jenersy. 1989.

Приложение 1. Примеры гибридных систем.

Рассмотрим несколько примеров простых гибридных систем, иллюстрирующих различные типы качественных скачков в поведении гибридных систем. На этих же примерах проиллюстрируем основные черты трех различных подходов к моделированию гибридных систем: блочное моделирование (Simulink), «физическое» моделирование (Modelica, Dymola) и использование гибридного автомата (Model Vision Studium).

Пример 1: прыгающий мячик.

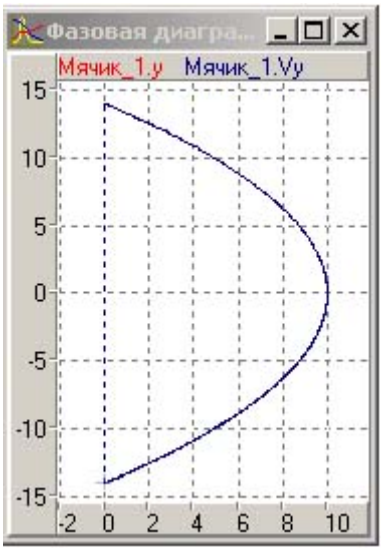


Рис. 78

Абсолютно упругий мячик (материальная точка) отпускается на высоте H над абсолютно твердой горизонтальной плоскостью. Движение описывается системой уравнений (1)

$$(1)\ \begin{cases} \dfrac{dy}{dt} = V_y \\ \dfrac{dV_y}{dt} = -g \end{cases} \qquad y(0) = H, \quad V_y(0) = 0$$

где y - вертикальная координата мячика относительно плоскости, V_y - вертикальная скорость мячика. Выход за область «Полет в поле тяготения» определяется предикатом $P(y, V_y) = (y \le 0)\, and\, (V_y \le 0)$. В точке переключения t^* скорость V_y скачком меняет знак: $V_y(t^* + 0) = -V_y(t^*)$, после чего движение мячика

снова описывается системой уравнений (1), но уже с новыми начальными условиями. Таким образом, этот пример иллюстрирует гибридное поведение первого типа. На Рис. 78 показана фазовая диаграмма этой системы для $H = 10$.

Пример 1 в подсистеме Simulink пакета MATLAB.

Этот пример, как типовой, поставляется в качестве демонстрационного примера для подсистемы Simulink, мы только изменили начальные условия. Блок-схема для примера 1 приведена на Рис. 79.

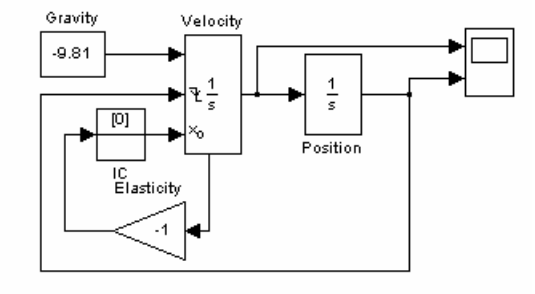


Рис. 79

При первом взгляде на эту блок-схему совершенно непонятно, каким образом все это работает. При внимательном рассмотрении выясняется, что все дело в хитром использовании многочисленных дополнительных возможностей класса Integrator (блок Velocity), а именно входа внешнего начального значения и входа сброса интегратора. На основной вход первого интегратора Velocity поступает константа –9.81 (ускорение силы тяжести). Начальное значение интегрируемой величины (скорости) берется с выхода блока IC класса InitialCondition. Этот блок устроен так, что в нулевой момент времени значением его выхода является значение параметра (в данном случае 0), а последующие моменты – значение входа. Значение скорости поступает в качестве входа на второй интегратор Position и на блок рисования графика. На выходе второго интегратора вычисляется текущее положение мячика, которое поступает на блок рисования графика и на вход сброса Reset первого интегратора Velocity. Значение скорости с дополнительного выхода StatePort (почему не с основного выхода – особенности численной реализации) через

усилитель Elactity с коэффициентом усиления –1 поступает на вход блока IC. Поскольку в установках блока Velocity указан режим работы входа Reset по заднему фронту сигнала сброса, то в момент, когда положение мячика переходит из положительного значения в 0, первый интегратор сбрасывается, величина скорости приобретает внешнее начальное значение (последнее значение скорости с обратным знаком) и интегрирование продолжается дальше, Следует отметить, что в данном случае в силу специфических особенностей стандартных блоков Simulink не нужно проверять условие Vy<0, так как сброса не произойдет даже если начальное положение мячика нулевое.

Прииер 1 на языке Modelica.

Описание этого примера на языке Modelica включает в себя задание константы g и переменных y,Vy с соответствующими начальными значениями, задание двух дифференциальных уравнений и дискретного события с соответствующим блоком when, в котором непрерывная переменная Vy заново инициализируется своим инвертированным значением.

```
model BauncingBall
  constant Real g = 9.81;
  Real Vy (start=0);
  Real y (start=10);
equations
  when (y<=0) and (Vy<0) then
    reinit(Vy,-Vy);
  end when;
  der(Vy) = -g;
  der(y)  = Vy;
end BauncingBall;
```

Здесь и далее уравнения даются в стандартном текстовом представлении языка Modelica. В то же время в конкретных пакетах, поддерживающих этот язык, уравнения могут представляться и в естественной математической форме.

Пример 1 в пакете Model Vision Studium.

Описание этого примера в пакете Model Vision Studium включает в себя определение переменных и константы

```
constant g: double := 9.81;
Vy: double := 0;
y: double := 0;
```

, систему уравнений «Система_уравнений_1» на Рис. 80а (системы уравнений здесь и далее представлены в форме, в которой они представляются в специальном редакторе уравнений, входящем в состав пакета) и гибридную карту состояний на Рис. 80б.

$$\frac{d\,y}{d\,t} = Vy$$

$$\frac{d\,Vy}{d\,t} = -g$$

а) б)

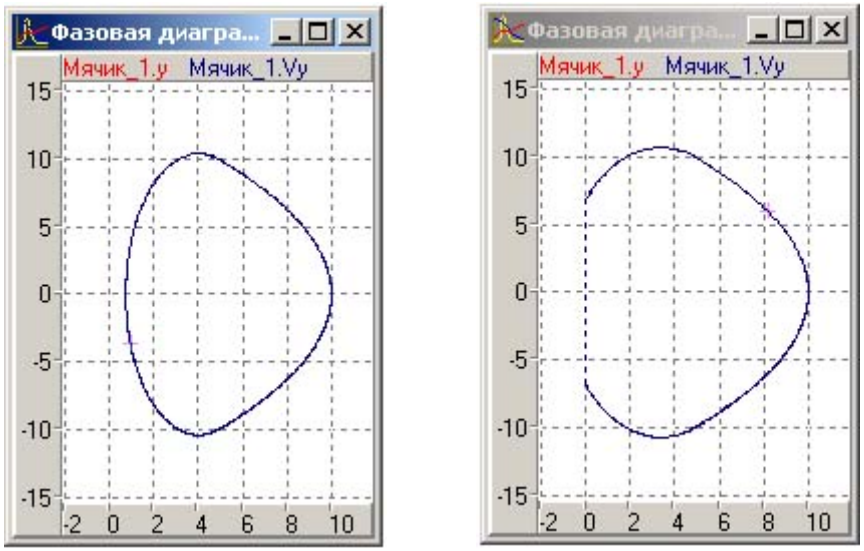
Рис. 80

Пример 2: мячик, падающий на пружину.

Пусть теперь мячик падает на свободный конец невесомой пружины с жесткостью K и длиной $H_s \le H$, закрепленной вертикально на плоскости. Движение мячика задается системой уравнений (1) при $y > H_s$ и системой уравнений (1а) при $y \le H_s$:

$$\text{(1а)} \begin{cases} \dfrac{dy}{dt} = V_y \\ \dfrac{dV_y}{dt} = K \cdot (H_s - y) - g \end{cases}$$

В зависимости от значения коэффициента жесткости пружины удара мячика о плоскость может не происходить (Рис. 81а) или происходить (Рис. 81б). Во втором случае в системе происходят как скачок типа (2), так и скачок типа (1).



а) б)

Рис. 81

Пример 2 в подсистеме Simulink пакета MATLAB.

Блок-схема для этого примера показана на Рис. 82.

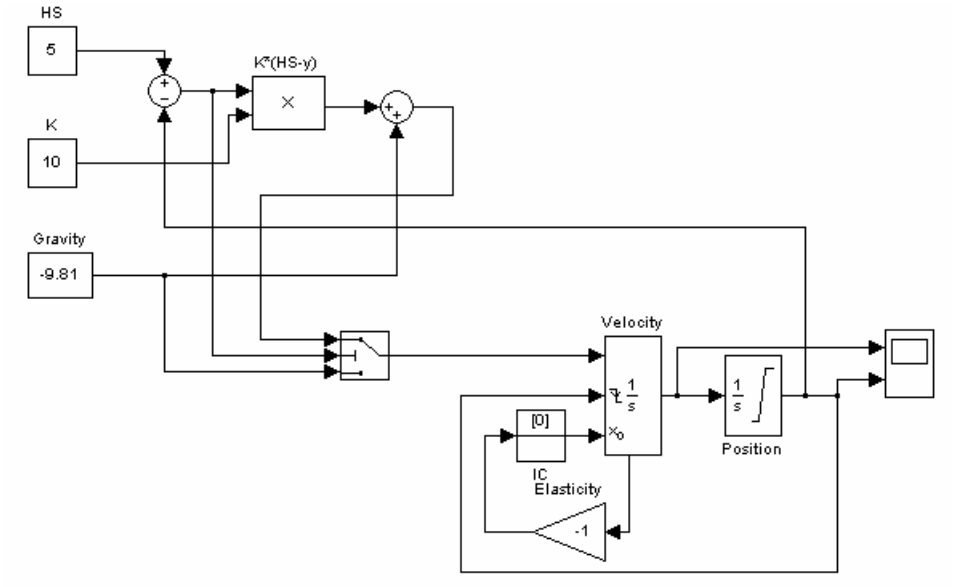


Рис. 82

Эта схема представляет собой модификацию блок-схемы, показанной на Рис. 79. На основной вход первого интегратора теперь подается не константа, а выражение

```
if (HS-y)>=0 then K*(HS-y)+(-g) else -g,
```

набранное из типовых блоков. Переключение ветвей условного выражения осуществляется с помощью блока Switch. Таким образом, по существу здесь используется условное уравнение, заданное в свойственной всем пакетам блочного моделирования затейливой форме.

Пример 2 на языке Modelica.

В данном примере вполне естественным является использование условного уравнения. Кроме того, вводятся две новые константы.

```
model BauncingBallWithString
  constant Real g = 9.81;
  constant Reak K = 10;
  constant Real HS = 5;
  Real Vy (start=0);
  Real y (start=10);
equations
  when (y<=0) and (Vy<0) then
    reinit(Vy,-Vy);
  end when;
  der(Vy) = if y>HS then -g else K*(HS-y)-g;
  der(y)  = Vy;
end BauncingBallWithString;
```

Пример 2 в пакете Model Vision Studium.

Здесь модификация примера 1 может проводиться двумя способами: 1) использованием условного уравнения; 2) использованием вложенной карты состояний. Кроме того, в обоих случаях необходимо ввести дополнительные параметры K и HS.

Использование условных уравнений.

Пакет Model Vision Studium наряду с гибридными картами состояний позволяет использовать и условные уравнения. В ряде случаев это приводит к значительно более компактному описанию. Заметим, что исполняющей системой пакета переключение ветвей условного уравнения трактуется как неявный переход в карте состояний и обрабатывается соответствующим образом.

В данном случае «Система_уравнений_1» должна быть модифицирована как показано на Рис. 83. Карта состояний в этом случае не меняется.

$$\frac{d\,y}{d\,t} = Vy$$

$$\frac{d\,Vy}{d\,t} = \left(\textbf{if } y>HS \textbf{ then } -g \textbf{ else } K\cdot\left(HS-y\right)-g\right)$$

Рис. 83

Использование вложенной карты состояний.

В этом случае «Система_уравнений_1» не изменяется (мы только переименум ее в «УравнениеСвободногоДвижения»). Дополнительно создадим систему уравнений «УравнениеДвиженияСПружиной» (Рис. 84).

$$\frac{d\,y}{d\,t} = Vy$$

$$\frac{d\,Vy}{d\,t} = -g+K\cdot\left(HS-y\right)$$

Рис. 84

В главной карте состояний уберем систему уравнений из состояния «Полет» и припишем ему другую карту состояний «ДвиженияМячика» (Рис. 85).

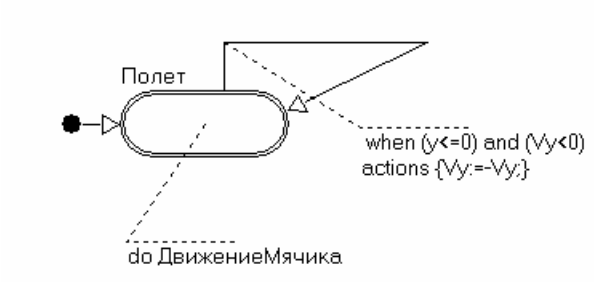


Рис. 85

В свою очередь карта состояний «ДвижениеМячика» определяет условия перключения между двумя системами уравнний (Рис. 86).

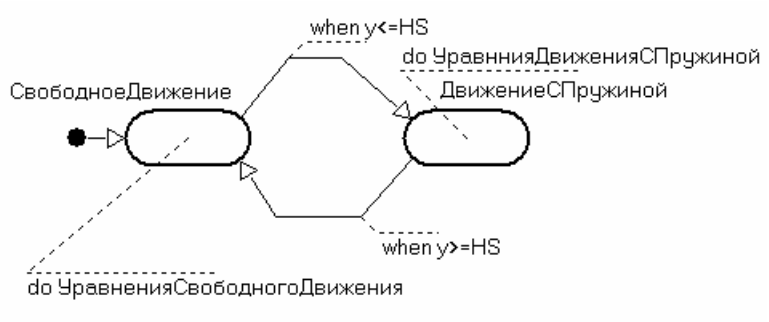


Рис. 86

Таким образом, мы разделили описания сложного движения мячика на две части: главная карта состояний описывает отскок от плоскости, а локальная карта состояний описывает взаимодействие с пружиной. Этот вариант несколько более громоздкий, чем вариант с условным уравнением, однако, он значительно более нагляден при отладке модели, поскольку переключения состояний можно наблюдать визуально и задавать останов модели по срабатыванию перехода.

Пример 3: отрывающийся маятник.

Моделируемая система представляет собой материальную точку (мы будем представлять ее как шарик достаточно малого размера), прикрепленную к нерастяжимому и невесомому стержню длиной L, другой конец которого шарнирно закреплен в начале системы координат (см. Рис. 87).

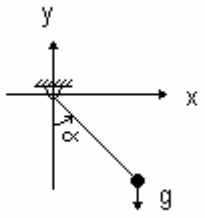


Рис. 87

Состояние маятника полностью определяется значением двух переменных: угла отклонения α и угловой скоростью ω.

Динамика маятника определяется двумя дифференциальными уравнениями:

$$(2)\ \begin{cases} \dfrac{d\alpha}{dt} = \omega \\ \dfrac{d\omega}{dt} = \dfrac{-g \cdot \sin\alpha}{L} \end{cases},\ \text{где}\ \alpha(0) = \alpha_0,\ \omega(0) = \omega_0 .$$

Пусть в некоторый момент t^* (например, определяемый условием $\alpha \geq \alpha_{\max}$) крепление шарика к стержню разрушается и далее шарик продолжает свое независимое от стержня движение. Движение шарика после отрыва задается системой уравнений:

$$(3)\ \begin{cases} \dfrac{dx}{dt} = V_x \\ \dfrac{dy}{dt} = V_y \\ \dfrac{dV_y}{dt} = -g \end{cases} .$$

Начальные условия для системы уравнений (3) вычисляются в момент t^* по формулам:

$$\begin{cases} x = L \cdot \sin\alpha \\ y = -L \cdot \cos\alpha \\ V_x = \omega \cdot \cos\alpha \\ V_y = \omega \cdot \sin\alpha \end{cases}$$

Рис. 88

На Рис. 88 показана траектория движения маятника для значений параметров $\alpha_0 = -\dfrac{\pi}{2}, \omega_0 = 0, \alpha_{\max} = \dfrac{\pi}{4}$.

Пример 3 в подсистеме Simulink пакета MATLAB.

На этот пример автор потратил более двух часов: очень уж неудобно набирать уравнения из блоков. Общая идея модели состоит в том, что собираются две подсистемы, соответствующие уравнениям колебаний и свободного полета, которые включаются и отключаются в зависимости от условия.

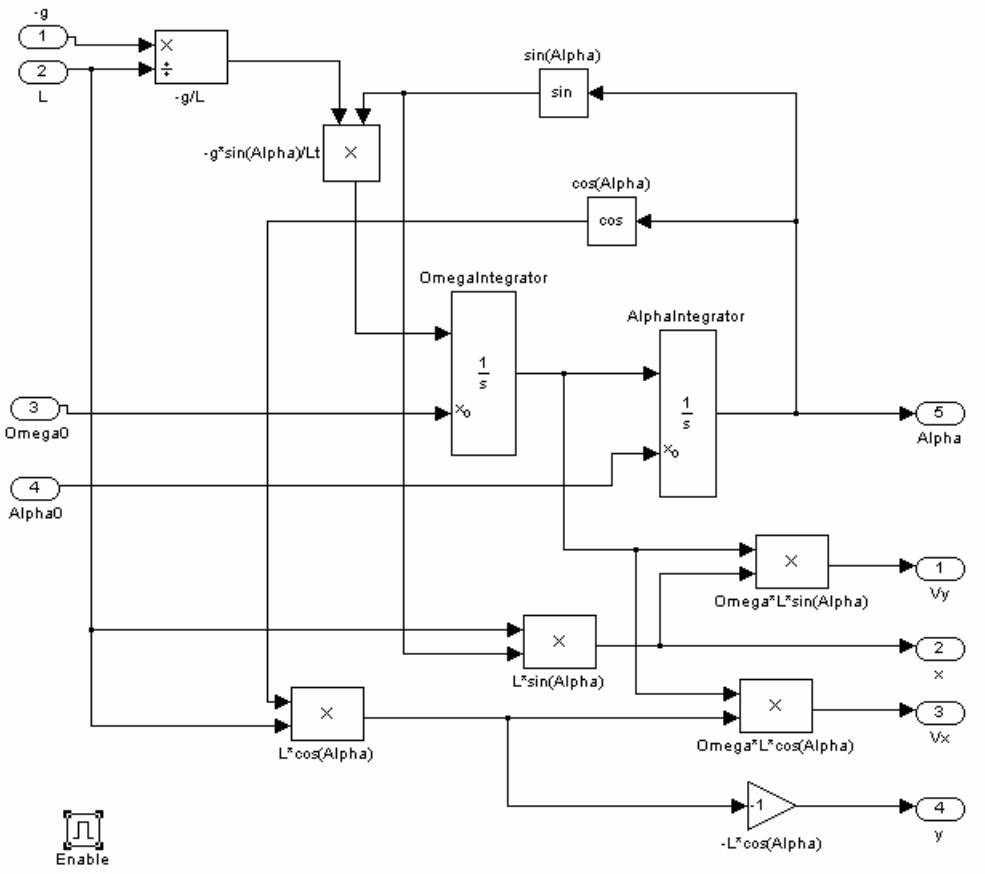


Рис. 89

На Рис. 89 показана блок-схема подсистемы Oscillations, соответствующей режиму колебаний маятника. Вместе с величинами α и ω в подсистеме вычисляются также и значения линейных скоростей и координат. Подсистема содержит блок класса Enable, который позволяет отключать всю подсистему по отрицательному значению его входного сигнала. В этом блоке имеется также важный параметр, который указывает, что делать с выходными значениями подсистемы после ее отключения: сбрасывать в начальные или удерживать значения на момент отключения. В данной подсистем установлен второй режим – удержание последних значений. Это необходимо для

инициализации подсистемы Flight. Все интеграторы в подсистеме имеют внешние начальные значения.

На Рис. 90 показана блок-схема подсистемы Flight, соответствующей уравнениям свободного полета.

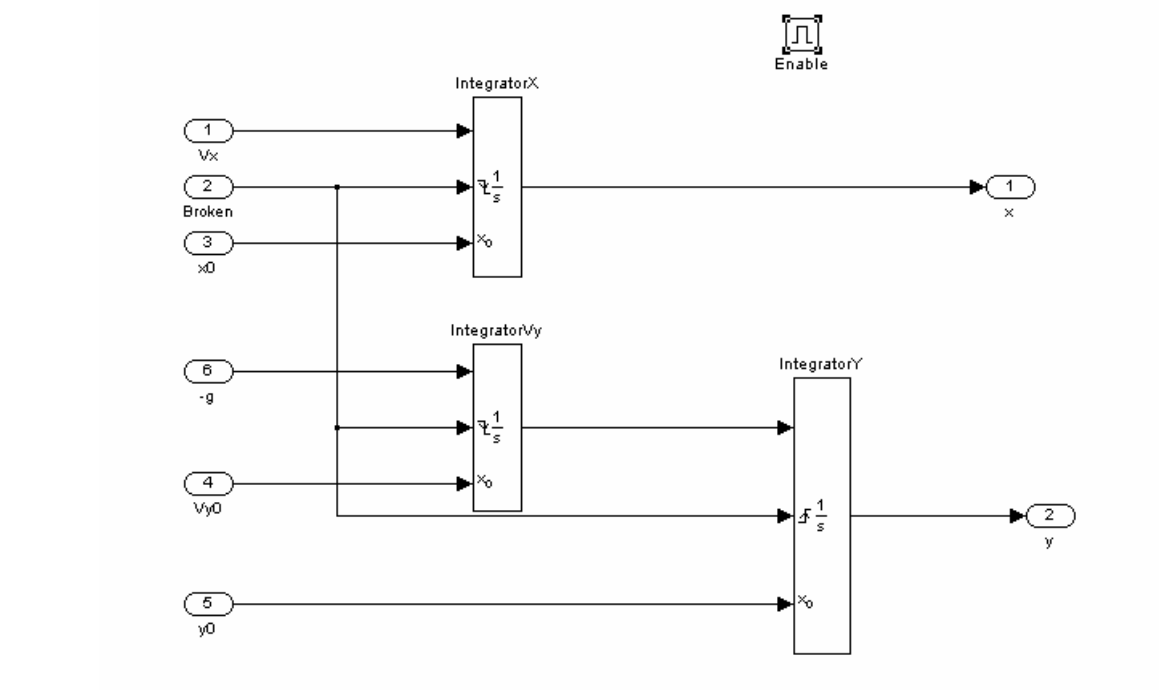


Рис. 90

В этой подсистеме также имеется блок Enable, а интеграторы имеют помимо внешних начальных значений интегрируемой величины еще и дополнительные входы сброса, которые управляются внешним сигналом Broken. В параметрах интеграторов указано, что сброс производится по положительному значению сигнала.

Наконец, на Рис. 91 приведена блок-схема модели в целом.

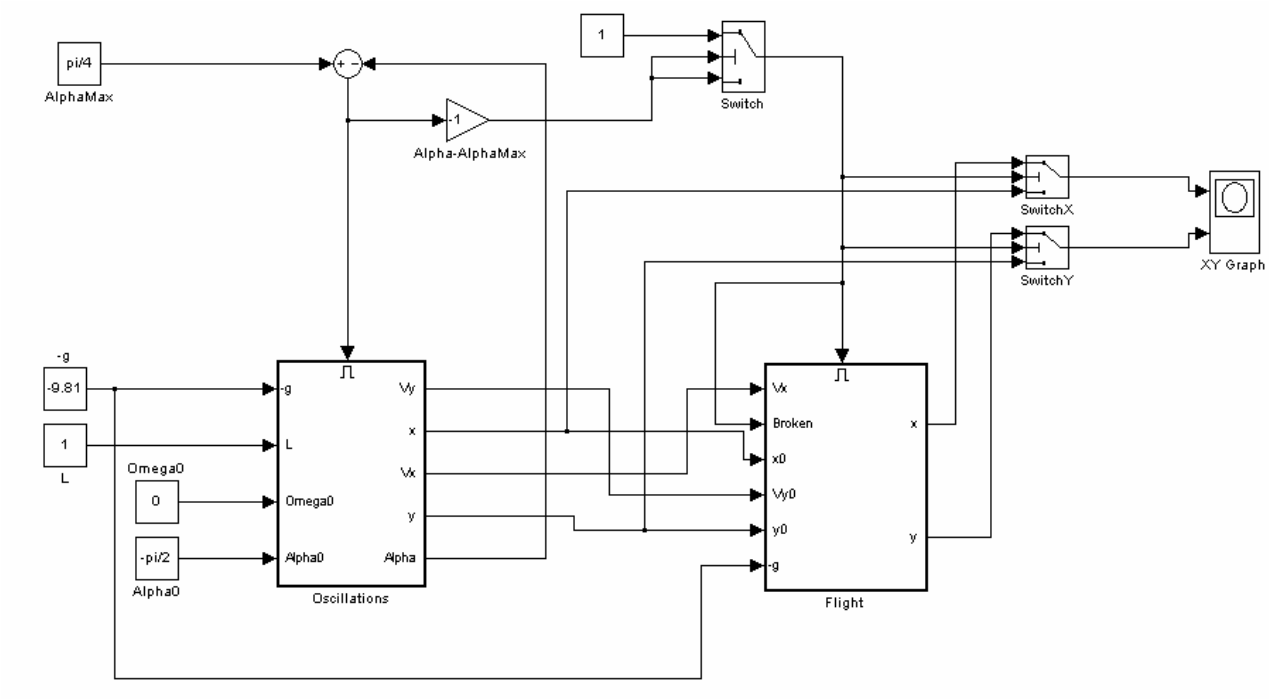


Рис. 91

Эта блок-схема помимо подсистем Oscillations и Flight содержит блоки задания констант, блок построения фазовой диаграммы для получения траектории движения в декартовых координатах и схему коммутации подсистем. Для коммутации подсистем вычисляется значение $\alpha_{\max} - \alpha$, которое подается на вход Enable подсистемы Oscillations. Таким образом, в момент $\alpha = \alpha_{\max}$ эта подсистема выключается, а ее выходы сохраняют последние вычисленные значения. Одновременно этот сигнал после инверсии (то есть $\alpha - \alpha_{\max}$) подается на управляющий вход специального переключателя, который при нулевом значении управляющего сигнала (то есть в момент $\alpha = \alpha_{\max}$) вырабатывает на выходе значение +1. Это значение подается на входы Enable и Broken подсистемы Flight. Оно же используется для коммутации сигналов, подаваемых на вход блока построения фазовой диаграммы.

Пример 3 на языке Modelica.

С этим примером язык Modelica справляется уже значительно хуже, чем с предыдущими двумя. Хотя в языке и предусматриваются системы уравнений, структура которых зависит от условия, это условие обязательно

должно быть статическим, то есть окончательная структура системы уравнений должна определяться на этапе компиляции модели, а не на этапе выполнения. В противном случае невозможно построить одноуровневую эквивалентную систему гибридных уравнений. Поэтому в руководстве по языку Modelica предлагается в такого рода случаях строить для каждого варианта системы уравнений свой уникальный объект, помещать экземпляры всех этих объектов в объект-контейнер. Затем по соответствующему дискретному событию нужно активизировать соответствующий объект-поведение, задать начальные значения его переменных и использовать далее выходные значения этого объекта как выходные значения всей системы. Для данного примера этот подход выливается в следующие действия.

Создаются отдельные модели колеблющегося и оторвавшегося маятника.

```
partial model BasePendulum // абстрактный класс
  parameter Real L;
  constant Real g = 9.81;
  output Real x,y,Vx,Vy;
end BasePendulum;
block Pendulum // колеблющийся маятник
  extends BasePendulum;
  output Real alpha (start=-pi/2), omega (start=0);
equations
  der(alpha)=omega;
  der(omega)=-g*sin(phi)/L;
  x=L*sin(alpha);
  y=-L*cos(alpha);
  Vx=omega*L*cos(alpha);
  Vy=omega*L*sin(alpha);
end Pendulum;
block BrokenPendulum // оторвавшийся маятник
  extends BasePendulum;
equations
  der(x)=Vx;
  der(y)=Vy;
  der(Vy)=-g;
```

```
end BrokenPendulum;
model BreaingPendulum // отрывающийся маятник
  extends BasePendulun (L=1);
  parameter Real phimax = pi/4;
protected
  Boolean Broken (start=false);
  Pendulum pend (L=L, enable=not Broken);
  BrokenPendulum bpend (L=L, enable-Broken);
equations
  when pend.phi>=phimax then
    Broken=true;
    reinit(bpend.x,pend.x);
    reinit(bpend.y,pend.y);
    reinit(bpend.Vx,pend.Vx);
    reinit(bpend.Vy,pend.Vy);
  end when;
  x = if Broken then bpend.x else pend.x;
  y = if Broken then bpend.y else pend.y;
  Vx = if Broken then bpend.Vx else pend.Vx;
  Vy = if Broken then bpend.Vy else pend.Vy;
end BreakingPendulum;
```

Таким образом, в данном примере Modelica предлагает по существу ту же самую схему построения модели гибридной системы, что и Simulink: для каждого поведения строится своя цепочка блоков, входы и выходы которой коммутируются в зависимости от дискретных событий. Отличием Modelica является лишь то, что уравнения можно просто и удобно записывать в исходном математическом виде вместо того, чтобы набирать их из набора блоков низкого уровня (сумматоров, интеграторов и т.п.).

Пример 3 в пакете Model Vision Studium.

Декларация переменных:

```
  parameter L: double := 1;
  parameter Ymin: double := -3;
  parameter AlphaMax: double := pi/4;
  Omega: double := 0;
  Alpha: double := -pi/2;
```

```
x: double := L*sin(Alpha);
y: double := -L*cos(Alpha);
Vx: double := 0;
Vy: double := 0;
x1: double := 0;
y1: double := 0;
constant g: double := 9.8066;
```

Качественное поведение задается гибридной картой состояний, показанной на Рис. 92.

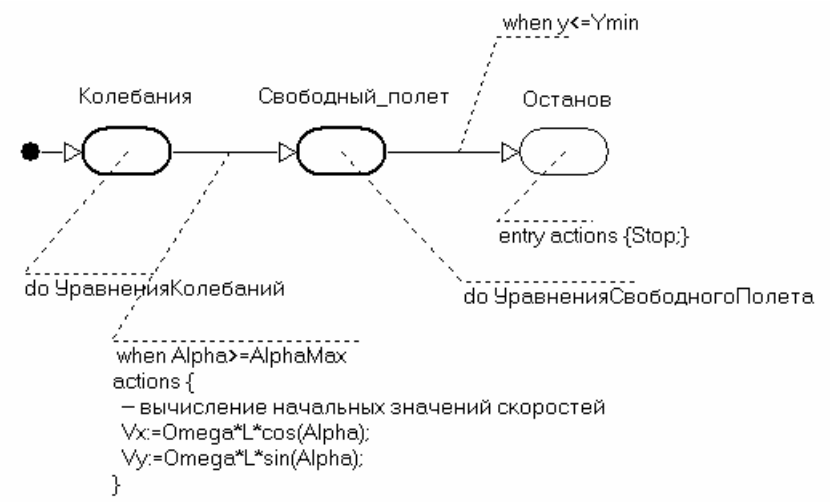


Рис. 92

Система уравнений для режима колебаний («УравненияКолебаний» показана на Рис. 93а, а система уравнений для режима свободного полета («УравненияСвободногоПолета») на Рис. 93б.

$$\frac{d\,\mathrm{Alpha}}{d\,t} = \mathrm{Omega}$$

$$\frac{d\,\mathrm{Omega}}{d\,t} = -g\cdot\frac{\sin(\mathrm{Alpha})}{L}$$

$$x = L\cdot\sin(\mathrm{Alpha})$$

$$y = -L\cdot\cos(\mathrm{Alpha})$$

а)

$$\frac{d\,x}{d\,t} = Vx$$

$$\frac{d\,y}{d\,t} = Vy$$

$$\frac{d\,Vy}{d\,t} = -g$$

б)

Рис. 93

Пример 4: выпрямитель.

На Рис. 94 изображена схема простейшего выпрямителя [113].

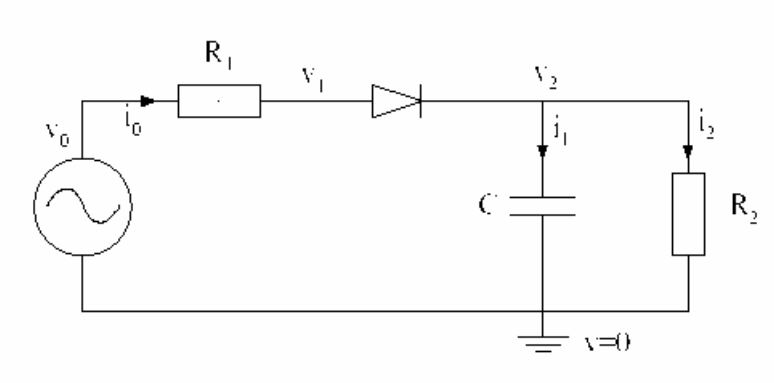


Рис. 94

Ее функционирование задается уравнениями («базовая система уравнений»):

$$\begin{cases} \dfrac{dV_2}{dt} = I_2 \\ I_0 = I_1 + I_2 \\ I_2 = \dfrac{V_2}{R_2} \\ R_1 \cdot I_0 = V_0 - V_1 \\ U = V_1 - V_2 \\ V_0 = A \cdot \sin 2\pi\omega t \end{cases}$$

Функционирование диода задается вольт-амперной характеристикой, показанной на Рис. 95 (а – реальная характеристика, б – идеальная).

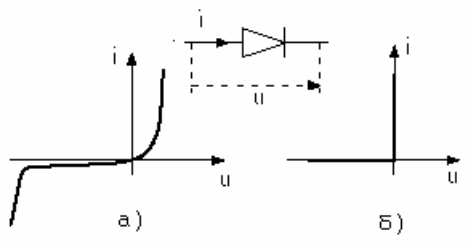


Рис. 95

Пример 4 на языке Modelica.

Поскольку идеальная характеристика диода никак не укладывается в рамки явного представления в форме уравнения, ее задают в параметрической форме с помощью вспомогательной переменной s [113].

```
model Rectifier
  parameter Real A = 220;
  parameter Real Omega = 50;
  parameter Real R1 = 100;
  parameter Real R2 = 100;
  parameter Real C = 1E-10;
  Real V0,V1,V2,U;
  Real I0,I1,I2;
  Real s;
  Boolean off;
equations
  off = (s<0);
  U = V1-V2;
  U = if off then s else 0;
  I0 = if off then 0 else s;
  R1*I0 = V0-V1;
  V0 = A*sin(2*pi*Omega*time);
  I2 = V2/R2;
  I1 = I0-I2;
  der(V2) = I1/C;
end Rectifier;
```

Пример 4 в пакете Model Vision Studium.

Поведение выпрямителя задается гибридной картой состояний

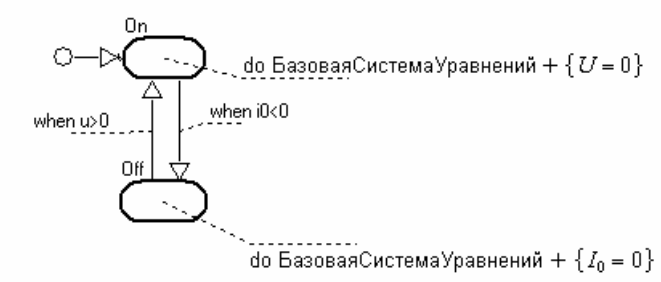


Рис. 96

Пример 5. Синхронная и асинхронная композиция гибридных автоматов.

Рассмотрим систему, показанную на Рис. 97. Эта система включает в себя два сумматора и два периодических прерывателя.

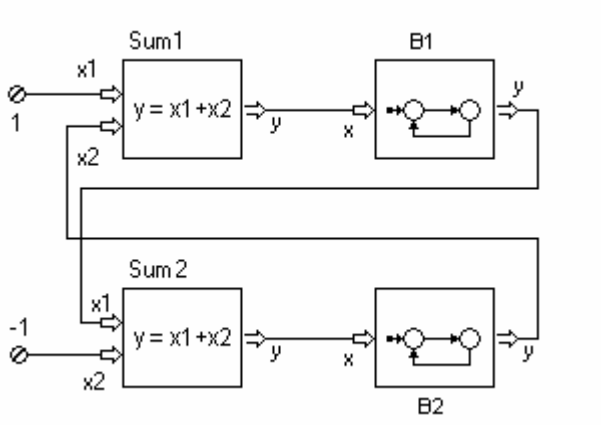


Рис. 97

Прерыватель с периодом T мгновенно передает значение входа x на выход y (см. Рис. 98).

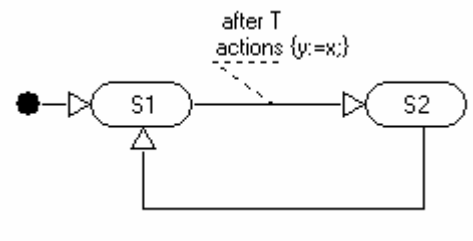
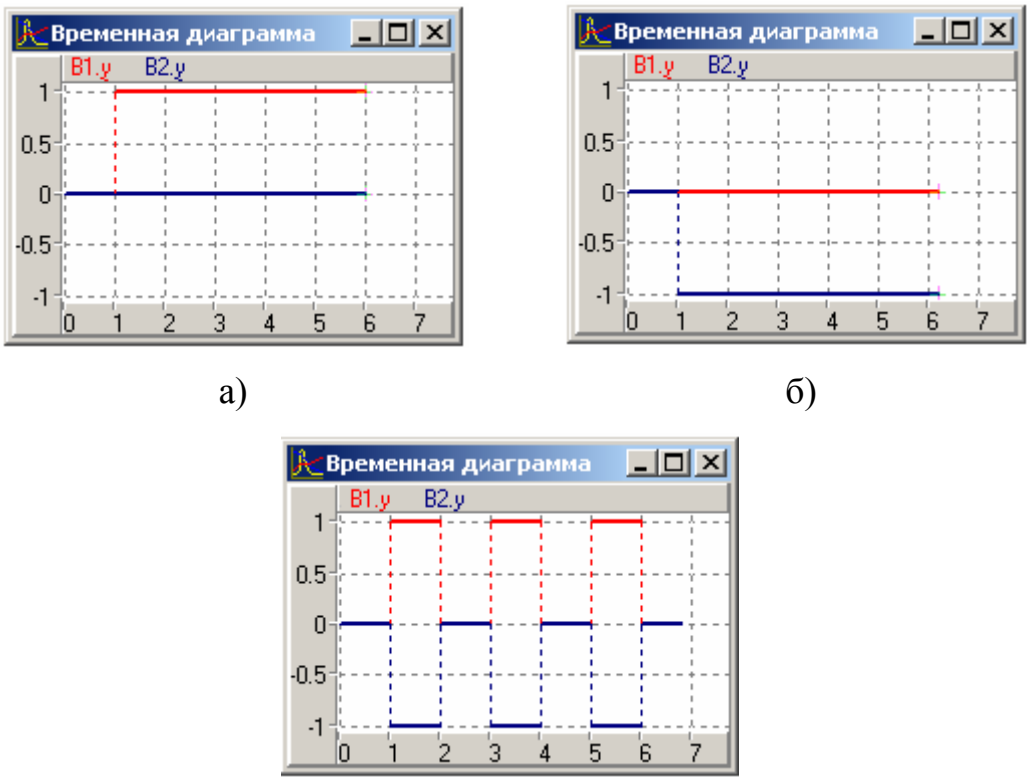


Рис. 98

Если на входы системы подать указанные значения, то при $B1.T < B2.T$ схема через время $B1.T$ переходит в устойчивое состояние $B1.y = 1, B2.y = 0$ (Рис. 99а), а при $B1.T > B2.T$ схема через время $B2.T$ переходит в устойчивое состояние $B1.y = 0, B2.y = -1$ (Рис. 99б). Эти зависимости справедливы как для синхронного, так и для асинхронного объединения. Однако, при $B1.T = B2.T$ мы получаем качественно различные поведения системы:

- при асинхронном объединении система по-прежнему переходит в одно из устойчивых состояний в зависимости от случайного порядка срабатывания переходов;

- при синхронном объединении наблюдаются колебания с периодом $2 \cdot B1.T$ (Рис. 99в).



а) б)

в)

Рис. 99

Попробуем разобраться, какое из этих поведений правильное. В рассматриваемой системе имеются два идеальных допущения: 1) замыкание цепи и передача значения входа на выход прерывателя происходит мгновенно; 2) сумматор вычисляет сумму мгновенно. Модифицируем карту поведений прерывателя: пусть теперь в течение ΔT он ведет себя как непрерывная система с уравнением $y = x$ (Рис. 100).

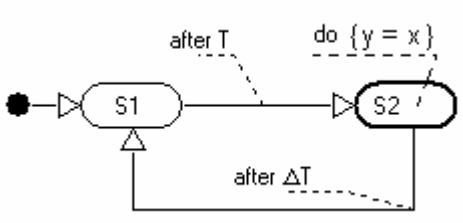


Рис. 100

Заметим, что при $\Delta T = 0$ такая карта поведений имитирует обработчик дискретных событий языка Modelica: во временной щели решается система уравнений, включающая уравнения сумматоров и «мгновенные» уравнения замыкания. Однако, в данном примере такая система уравнений оказывается вырожденной и имеет бесконечно много решений. Если же мы заменим уравнение $y = x$ на уравнение $y = pre(x)$, то получим колебания, показанные на Рис. 99в.

Чтобы убрать второе идеальное допущение, вставим после сумматоров апериодические звенья (Рис. 101) с постоянной времени T_A. При $\Delta T \to 0,\ \ T_A \to 0$ мы должны получить предельный случай, соответствующий идеальному.

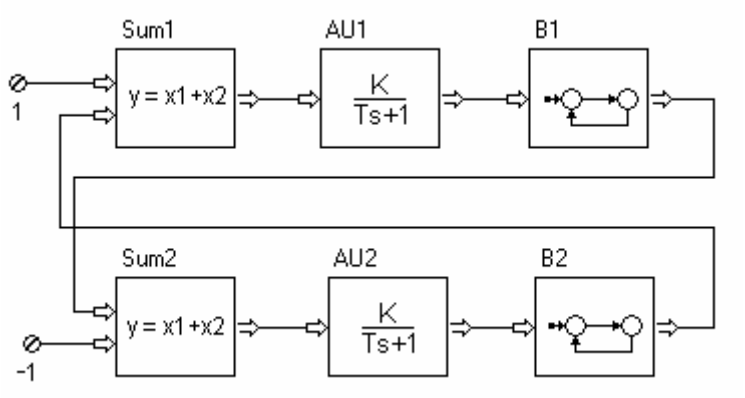
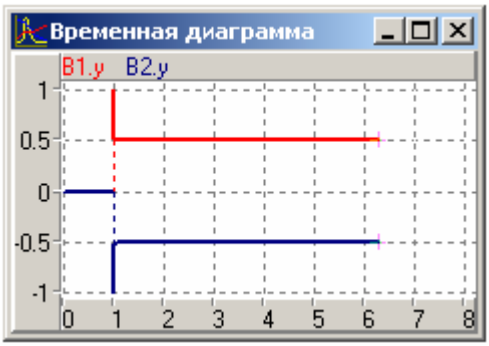


Рис. 101

На Рис. 102 показаны зависимости выходов от времени при значениях ΔT 0.1 (а), 0.05 (б), 0.001 (в), 0 (г) (во всех случаях $T_A = 0.001$). Таким образом, поведение синхронного гибридного автомата действительно соответствует предельному случаю поведения неидеальной системы.


Временная диаграмма
B1.y B2.y
1
0.5
0
-0.5
-1
0 1 2 3 4 5 6 7 8

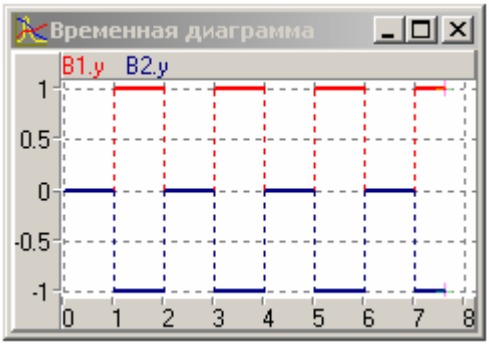
а)


Временная диаграмма
B1.y B2.y
1
0.5
0
-0.5
-1
0 1 2 3 4 5 6 7 8

б)


Временная диаграмма
B1.y B2.y
1
0.5
0
-0.5
-1
0 1 2 3 4 5 6 7 8

в)


Временная диаграмма
B1.y B2.y
1
0.5
0
-0.5
-1
0 1 2 3 4 5 6 7 8

г)

Рис. 102

Приложение 2. Примеры моделей на языке MVL.

В данном Приложении подобно разобраны некоторые модели, иллюстрирующие основные понятия языка MVL.

Пример 1. Синхронизация с помощью сигнала: часы с боем.

Пусть на крепостной башне имеются часы, которые показывают текущее время с помощью часовой и минутной стрелок, а также отбивают часы. Пусть на башне также имеется дежурный наблюдатель, который в полдень и полночь по двенадцатому удару часов стреляет из пушки. На Рис. 103 показана блок-схема модели, включающая два гибридных автомата – «Часы» и «наблюдатель». Блок «Часы» имеет два выхода вещественного типа (угловые положения минутной и часовой стрелок Phi_M, Phi_H), а также выход «Удар» типа *signal*. Блок «Наблюдатель» имеет вход «Удар» и выход «Выстрел» типа *signal*.

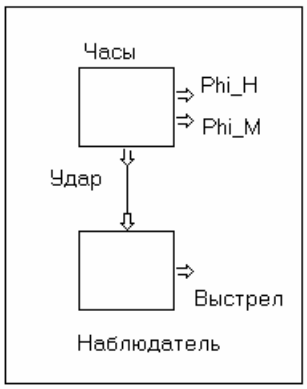


Рис. 103

На Рис. 104 показан граф переходов блока «Часы».

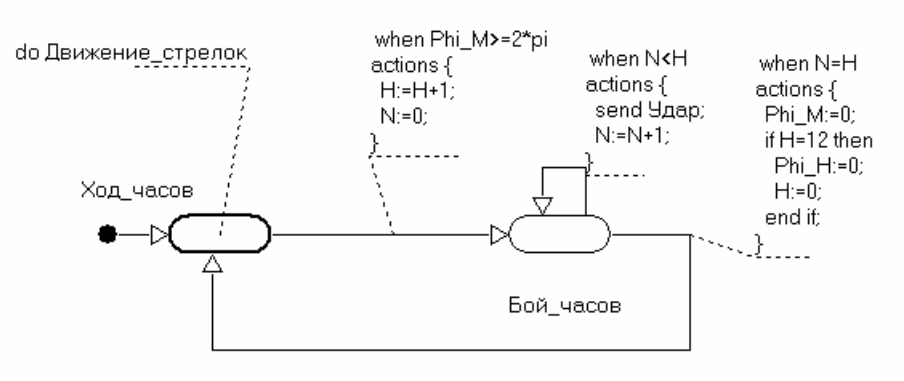


Рис. 104

В состоянии «Ход_часов» решается система уравнений «Движение_стрелок»

$$\begin{cases} \dfrac{dPhi\_M}{dt} = \dfrac{2\pi}{1} \\ \dfrac{dPhi\_H}{dt} = \dfrac{2\pi}{12} \end{cases}$$ (предполагается, что модельное время отсчитывается в часах).

При завершении минутной стрелкой полного круга число истекших часов H увеличивается на единицу и далее во временной щели происходит «бой часов»: H раз генерируется сигнал «Удар». После этого временная щель завершается и снова продолжается плавное движение стрелок. На Рис. 105 показан граф переходов блока «Наблюдатель».

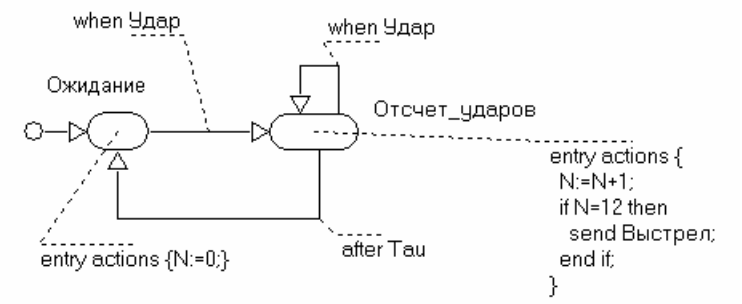


Рис. 105

По первому сигналу «Удар» автомат переходит из состояния «Ожидание» в состояние «Отсчет_ударов», в котором и остается до окончания временной щели. По 12-му удару выдается сигнал «Выстрел». Возвращение в состояние «Ожидание» производится по истечении любого интервала времени $0 < \tau < 1$, свидетельствующего о завершении серии ударов часов.

Пример 2. Компонентная модель отрывающегося маятника.

Рассмотрим более подробно модель отрывающегося маятника (пример 3 в Приложении 1), уделяя особое внимание аспектам ООМ. Качественная динамика отрывающегося маятника задается картой состояний, показанной на Рис. 106.

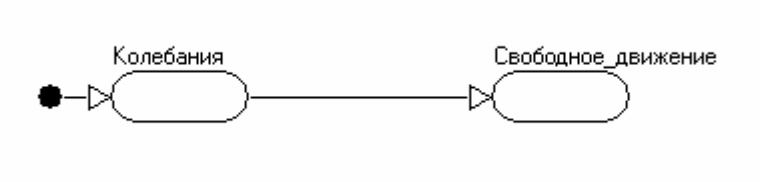


Рис. 106

Для полного определения этой модели необходимо задать деятельности в двух ее состояниях, а также условие срабатывания и мгновенные действия в переходе. Модель отрывающегося маятника должна содержать две переменные, отражающие текущее положение материальной точки и параметры, задающие длину маятника, начальный угол отклонения маятника и угол обрыва:

```
x: double;
y: double;
parameter L: double := 1;
parameter Alpha_0: double := -pi/2;
parameter Alpha_B: double := pi/4;
```

Вариант 1. Использование локальных классов.

В этом варианте деятельности в различных состояниях взаимодействуют через общие переменные. В качестве деятельностей используем экземпляры двух локальных классов, определяющих непрерывные системы, которые отражают динамику модели в двух ее состояниях и оперируют набором общих переменных. Эти класса можно задать явно и присвоить им имена, а можно задать неявно, просто редактируя определение деятельности в соответствующем состоянии. Именованный локальный класс можно повторно использовать как деятельность или элемент деятельности в других состояни-

ях (например, в модели выпрямителя, рассмотренной в Приложении 1, деятельности в двух состояниях являются результатом объединения базовой системой уравнений с дополнительным уравнением, которое различно в разных состояниях). В этом варианте набор переменных модели следует дополнить переменными

```
Omega: double := 0;
Alpha: double := -pi/2;
Vx: double;
Vy: double;
```

Локальный класс `Маятник` задается системой уравнений

$$\begin{cases} \dfrac{dAlpha}{dt} = Omega \\ \dfrac{dOmega}{dt} = -g\dfrac{\sin(Alpha)}{L} \\ x = L\cdot\cos(Alpha) \\ y = -L\cdot\sin(Alpha) \end{cases}$$

Локальный класс `ДвижениеXY` задается системой уравнений

$$\begin{cases} \dfrac{dx}{dt} = Vx \\ \dfrac{dy}{dt} = Vy \\ \dfrac{dVy}{dt} = -g \end{cases}$$

Переход между состояниями задается соотношением

```
when (Alpha>=Alpha_B) / {  Vx := Omega*L*cos(Alpha);
                           Vy := Omega*L*sin(Alpha);}
```

Предполагается, что константа g определена в пакете проекта.

Последние текущие значения переменных, соответствующих состоянию колебаний, используются для вычисления начальных значений переменных, соответствующих свободному движению. Данный вариант является наиболее простым и понятным, однако, существуют случаи, когда нельзя использовать общие переменные и локальные классы (например, управление экспериментом).

Вариант 2. Использование независимых классов.

Сначала мы создадим и отладим модель математического маятника. Это изолированная непрерывная система, описание которой задается набором переменных

```
parameter L: double := 1;
Omega: double := 0;
Alpha: double := -pi/2;
```

и системой уравнений

$$\begin{cases} \dfrac{dAlpha}{dt} = Omega \\ \dfrac{dOmega}{dt} = -g\dfrac{\sin(Alpha)}{L} \end{cases}$$

Сохраним эту модель как класс Маятник. Далее создадим и отладим модель двумерного движения материальной точки в поле тяготения. Это тоже изолированная непрерывная система, описание которой задается набором переменных

```
x:  double := 0;
y:  double := 0;
Vx: double := 0;
Vy: double := 0;
```

и системой уравнений

$$\begin{cases} \dfrac{dx}{dt} = Vx \\ \dfrac{dy}{dt} = Vy \\ \dfrac{dVy}{dt} = -g \end{cases}$$

В этой системе уравнений используется константа проекта

```
constant g: double := 9.8066; -- м/сек**2
```

Сохраним эту модель как класс ДвижениеXY.

Теперь начинаем создавать модель отрывающегося маятника. Нам нужно каким-то образом поместить в качестве деятельности в состояние Колебания экземпляр класса Маятник, а в состояние Свободное_движение – экземпляр класса ДвижениеXY. При этом необходимо

правильно установить для этих объектов начальные условия и связать их переменные с переменными модели. Кроме того, необходимо сформулировать условие срабатывания перехода, отражающего обрыв маятника. Проблемы заключаются в том, что независимые классы используют свои собственные наборы переменных, а деятельности – экземпляры этих классов – определены и видимы только в соответствующих состояниях.

Следовательно, необходимо строить объект-деятельность как композицию из объекта – экземпляра независимого класса и уравнений связи – экземпляра некоторого анонимного класса. Например, деятельность в состоянии Колебания можно определить так

```
P: Маятник := new Маятник (L:=L, Alpha:=Alpha_0,
Omega:=0) +
```

$$\begin{cases} x = L \cdot \sin(P.Alpha) \\ y = -L \cdot \cos(P.Alpha) \end{cases}$$

Теперь необходимо разобраться с условиями и действиями перехода. Запускающее событие («триггер») для перехода определяется условием

```
when (P.Alpha>=Alpha_B)
```

Экземпляр P класса Маятник виден только в описании состояния Колебания Поэтому возможны два решения:

1) ввести рабочую переменную карты состояний Alpha и добавить к определению деятельности в состоянии Колебания дополнительное уравнение

$$Alpha = P.Alpha\ .$$

В этом случае условие срабатывания перехода запишется как

```
when (Alpha>=Alpha_B)
```

2) использовать внутренний переход в состоянии Колебания, генерирующий дискретное событие

```
when (P.Alpha>=Alpha_B) / send Break(Alpha,Omega);
```

В последнем случае в карте состояний должна быть декларирована переменная

```
Break: signal (Alpha,Omega: double);
```

а условием внешнего перехода будет выражение

```
when Break.
```

В описании охраняющего условия и действий перехода будут видимы переменные `Break.Alpha, Break.Omega` типа `double`.

Наконец, остается только проблема вычисления начальных условий для линейных скоростей в состоянии `Свободное_движение`. Проблема собственно в том, что объект-деятельность в состоянии `Колебания` уже не существует в момент инициализации состояния `Свободное_движение`. Для этого необходимо ввести переменные `Vx0,Vy0` в карте состояний и присвоить им значение, например, в выходных действиях состояния `Колебания`:

```
Vx0 := P.L*P.Omega*cos(P.Alpha);
Vy0 := P.L*P.Omega*sin(P.Alpha);
```

либо – в случае использования сигнала - в действиях перехода

```
Vx0 := L*Break.Omega*cos(Break.Alpha);
Vy0 := L*Break.Omega*sin(Break.Alpha);
```

Теперь остается только определить деятельность в состоянии `Свободное_движение`:

```
M: ДвижениеXY := new ДвижениеXY(x:=x, y:=y, Vx:=Vx0, Vy:=Vy0)
+
```

$$\begin{cases} x = M.x \\ y = M.y \end{cases}$$

Пример 3. Наследование. Двумерное движение в воздухе.

Попробуем создать модель двумерного движения материальной точки в воздухе путем модификации созданного ранее класса `ДвижениеXY` . Прежде всего нам понадобятся дополнительные параметры: понадобятся новые параметры: коэффициент сопротивления C_x, площадь среднего сечения S_m , масса m (в среде с сопротивлением падение тела зависит от массы) и плотность воздуха ρ. Затем нам понадобится дополнительная переменная F для значения силы сопротивления воздуха, V. для значения полной скорости, $\sin\theta, \cos\theta$ для значений синуса и косинуса угла наклона траектории (угла тан-

гажа). После этого остается только переопределить уравнение $E3$ для Vy и добавить 5 новых уравнений:

$$\begin{cases} \frac{dx}{dt} = Vx \\ \frac{dy}{dt} = Vy \\ E3: \frac{dVy}{dt} = -g \end{cases} \rightarrow \begin{cases} \frac{dx}{dt} = Vx \\ \frac{dy}{dt} = Vy \\ E3: \frac{dVy}{dt} = -g - \frac{F \cdot \sin\theta}{m} \\ \frac{dVx}{dt} = -\frac{F \cdot \cos\theta}{m} \\ \sin\theta = \frac{Vy}{V} \\ \cos\theta = \frac{Vy}{V} \\ V = \sqrt{Vx^2 + Vy^2} \\ F = Cx \cdot Sm \cdot \frac{\rho \cdot V^2}{2} \end{cases}$$

Колесов Ю.Б.

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ МОДЕЛИРОВАНИЕ СЛОЖНЫХ ДИНАМИЧЕСКИХ СИСТЕМ

Лицензия ЛР № 020593 от 07.08.97

Налоговая льгота – Общероссийский классификатор продукции
ОК 005-93, т. 2; 95 3004 – научная и производственная литература

Подписано в печать 15.03.2004. Формат 60×84/16.
Усл. печ. л. 15,0 Уч.-изд. л. 15,0. Тираж 100. Заказ № 134.

Отпечатано с готового оригинал-макета в типографии Издательства СПбГПУ.
Санкт-Петербург, Политехническая, 29.
Отпечатано на ризографе RN-2000FP
Поставщик оборудования – фирма «Р-ПРИНТ»
Телефон: (812)110-65-09
Факс:(812)315-23-04