ИНФОРМАТИКА

2005 **январь-март** **№ 1**

УДК 519.71

Д.И. Черемисинов

МОРФИЗМЫ МОДЕЛЕЙ ПОВЕДЕНИЯ РАСПРЕДЕЛЕННЫХ СИСТЕМ

Анализируются ключевые формализмы теории распределенных систем: обобщенные автоматы, сети Петри, временная логика и язык ПРАЛУ. В результате анализа известных способов описания поведения вычислительных систем, построенных как набор взаимодействующих компонентов, показана перспективность языка ПРАЛУ в проектировании протоколов. Рассматриваются морфизмы интерпретации языков. Морфизм интерпретации часто используется, чтобы задать семантику языков описания процессов. По критерию богатства возможных интерпретаций язык ПРАЛУ можно считать более предпочтительным по сравнению с остальными.

Введение

Понятия параллелизма и асинхронности – это ключевые понятия, лежащие в основе современных представлений о проблемах проектирования вычислительных устройств. В работе [1] рассматриваются способы описания поведения агентов в ходе проектирования систем, особое внимание уделяется трактовке понятия времени в формализмах, моделирующих проектируемую систему. Показано, что во всех формализмах время учитывается как порядок смены событий, несмотря на отсутствие явных ссылок. Выбор формальных моделей и инструментов при проектировании распределенной системы представляет собой набор решений, которые могут иметь важные последствия для успешного окончания проекта. Известно большое число формальных моделей, которые естественным образом описывают распределенные системы. Эти модели имеют в основе математические формализмы, позволяющие описывать и анализировать поведение распределенных вычислительных систем. Формализм обеспечивает теорию для понимания поведения систем и методы проектирования и анализа.

Формальные модели обеспечивают основу для развития всех других теоретических и практических исследований распределенных вычислений. Большое количество предложенных за прошлые годы моделей требует их сравнительной оценки и классификации. Настоящая статья не претендует на полноту обзора предложенных моделей. Выбор нескольких ключевых языков описания распределенных систем и лежащих в их основе моделей основан на фундаментальности этих моделей в теории распределенных вычислений. В работе рассматриваются морфизмы *интерпретации* языков. Морфизм *интерпретации* часто используется, чтобы задать семантику языков описания процессов.

В математике морфизм – это абстракция функции или отображения. Морфизмы и пространства (или объекты), на которых они определены, образуют отрасль математики, которая называется теорией категорий [2]. В теории категорий морфизм рассматривается и изображается как стрелка между двумя различными объектами. Примером модели абстрактных морфизмов является теория конкретных категорий [2], в которой объекты – это просто множества с некоторой дополнительной структурой, а морфизмы представляют собой функции, сохраняющие эту структуру.

1. Отображение параллелизма и асинхронности в описании поведения

Стремление к минимальности средств, а также распространенное мнение о трудности понимания описаний, в которых присутствует параллелизм, обусловливают использование моделей протоколов, в которых понятия параллелизма и асинхронности не применяются. Однако модели, использующие параллелизм, позволяют составить представление о таких ошибках в поведении, как тупики и ловушки, и на основе этого научиться избегать их. Физические понятия параллелизма и асинхронности имеют много аспектов, не все из которых целесообразно отражать в математической модели. В известных моделях поведения дискретных устройств физический параллелизм компонентов отображается по-разному. Сформировавшееся мнение о

трудности анализа систем с учетом параллелизма в немалой степени объясняется тем, что существуют различия в понимании этого явления.

В работе [3] предлагается классификация моделей распределенных систем по трем параметрам: типу модели (поведенческая или структурная), типу параллелизма (чередование или истинный параллелизм), модели времени (линейная или ветвящаяся). Значения параметров предполагаются ортогональными, что приводит к восьми основным моделям. Взаимосвязи между моделями рассматриваются как морфизмы. Все эти модели используют в качестве базового понятие атомарного изменения – событие.

1.1. Обобщенный конечный автомат

Обобщенный конечный автомат [4] занимает лидирующее положение среди формализмов теории протоколов. Предпочтение этой модели объясняется использованием языков описания протоколов SDL и Estelle [5], которые базируются на понятии расширенного автомата, в качестве стандартных для описания протоколов в Международной организации по стандартизации (МОС). Этот формализм известен под различными наименованиями. Кроме обобщенного конечного автомата, используется также такой термин, как абстрактная машина данных и т. д. Сейчас более употребительно наименование «модель типа *состояние/переход*» (transition system) – тип структурных моделей по классификации [3].

Базовыми понятиями этого формализма являются переход, переменная и ее значение. Множество переменных определяет пространство состояний обобщенного автомата, а отдельное его состояние определяется значениями, присвоенными каждой переменной. Обычно одна из переменных выделена и называется основным состоянием, другие переменные называются контекстными (памятью абстрактной машины) [4]. Переменная основного состояния фиксирует текущее основное состояние. События в этой модели – это переходы. Переходы задаются из одного основного состояния в другое. Переходы зависят от текущего состояния и предикатов над контекстными переменными. С каждым переходом связан оператор, выполняемый во время перехода. Оператор задается функцией на множестве значений контекстных переменных. Результатом выполнения перехода (присоединенного оператора) является изменение значения контекстных переменных или взаимодействие с внешней средой. Значения контекстных переменных после каждого перехода определяются или вычислением операторов переходов, или входным воздействием на автомат. Конструкция, объединяющая основное состояние, предикат, задающий условие выполнения перехода, и оператор перехода в новое состояние, называется командой абстрактной машины. В языках описания протоколов, использующих эту модель, предикат и оператор команды задаются на одном из языков программирования. В SDL и Estelle эту роль выполняет Паскаль.

Формализм обобщенных конечных автоматов ориентирован на описание функций компонент распределенной системы – протокольных объектов и среды взаимодействия. Вся система в целом требует для своего описания более широкой модели – сети взаимодействующих автоматов. Включение в модель автомата произвольных переменных позволяет компактно представить целые группы переходов традиционного конечного автомата командами абстрактной машины в «интервальном» виде.

Понятие параллелизма в этом формализме не используется. Наибольшее значение модель обобщенных конечных автоматов имеет на этапе проверки соответствия документации и реализации протокола. Основное назначение формального описания в этот момент состоит в том, что оно служит эталоном внешнего поведения реализации. Описание протокола в виде обобщенного автомата в наибольшей степени отвечает указанной цели вследствие относительной малообъемности знаний, требуемых для использования формализма.

Чтобы показать правильность протокола, описанного расширенным автоматом, генерируется пространство состояний системы как декартово произведение пространств состояний протокольных компонент и среды взаимодействия. Затем выполняется анализ достижимости каждого состояния в ходе построения графа переходов автомата, заданного сетью компонентных автоматов системы. При таком анализе можно обнаружить тупики и недостижимые состояния.

Тупиком считается достижимое состояние, из которого нет ни одного перехода. Генерация пространства состояний и анализ достижимости могут быть легко автоматизированы. Та-

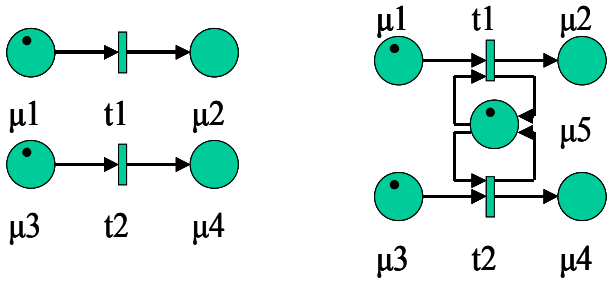
кая автоматическая верификация уменьшает вероятность того, что некоторые ситуации (комбинации состояний компонент) не будут учтены. Недостаток метода, делающий его впрямую неприменимым на практике, состоит в том, что по мере роста числа состояний компонентных автоматов число состояний всей системы экспоненциально возрастает. Это явление известно как «взрыв в пространстве состояний».

Основная возможность преодоления взрыва в пространстве состояний состоит в том, что в реальных системах огромное множество полных состояний системы может быть разбито на небольшое множество реально различимых классов состояний. Выделение этих классов требует привлечения понятий параллелизма и асинхронности.

В самой модели не присутствует понятие события. Возможна интерпретация этого формализма, когда неделимым атомарным изменением считается состояние. Различие в интерпретациях события легко обнаруживается путем моделирования. В результате моделирования получаются истории (трассы), состоящие из последовательностей переходов или состояний в зависимости от *интерпретации* модели. Очевидно, что эти истории тоже могут быть формально описаны. Соответствующие модели в классификации [3] называются поведенческими. Отображение структурной модели в поведенческую называется формальной семантикой. Так, для поведенческих моделей семантикой можно считать отображение в другую модель, например структурную. Для рассматриваемого формализма семантика зависит от *интерпретации* понятия события.

1.2. Сети Петри

Сети Петри являются самым распространенным в настоящее время формализмом, описывающим структуру и взаимодействие параллельных систем с асинхронным взаимодействием [6, 7]. В этом формализме возможен широкий спектр интерпретаций элементов модели как семантически, так и по уровню абстракции. Использование сетей Петри позволяет свести исследование протокола к изучению логической структуры сети, абстрагируясь от функций, которые должен реализовать протокол.



а) *б)*

Рис. 1. Примеры сетей Петри

Сеть Петри – это двудольный ориентированный граф с кратными дугами без петель, характеризующийся тремя параметрами П, Σ, С (рис. 1). Здесь П – это множество вершин, называемых позициями и обозначаемых кружками (квадратами); Σ – множество вершин, называемых переходами и обозначаемых черточками (полками); С – матрица инцидентности сети Петри. На множестве П задается функция, присваивающая каждой позиции некоторое целое неотрицательное число и называемая маркировкой сети. Число, присвоенное маркировкой некоторой позиции, изображается точками в кружке.

Сеть интерпретируется как динамическая модель, состояния которой задаются текущим значением маркировки. Маркировка разбивает множество переходов на два непересекающихся подмножества: возбужденных и невозбужденных переходов. Переход возбужден, если в каждой входной позиции число маркеров не меньше кратности входной дуги. Позиция называется входной (выходной) для перехода, если она соединена с ним дугой, направленной от позиции к переходу (от перехода к позиции). В начальный момент времени задается начальная маркиров-

ка, а затем в произвольном порядке, но только по одному, могут срабатывать возбужденные переходы. Срабатывание перехода ведет к изменению текущей маркировки и множества возбужденных переходов. Срабатывая, переход отбирает от каждой входной позиции некоторое количество маркеров, равное кратности связывающей дуги, и добавляет в каждую выходную позицию по порции маркеров, объем которой равен кратности связывающей дуги.

Интерпретация сетей Петри, обеспечивающая связь формализма с реальными системами, основывается на понятиях события и условия. Событие символизируется срабатыванием перехода сети. Условие возникновения события задается текущей маркировкой, причем маркировка указывает только часть условий, остальные остаются в модели незаданными, что может интерпретироваться как влияние внешней среды. В таком понимании отношение достижимости маркировок сети Петри задает систему причинно-следственных отношений между событиями.

1.2.1. Параллелизм в сети Петри. Сеть с заданной начальной маркировкой можно рассматривать как средство задания множества последовательностей срабатывания переходов, т. е. цепочек событий, каждое из которых можно представить точкой на оси времени. Таким образом, сеть Петри – это один из способов задания временных зависимостей между «точечными» событиями, причем время в такой интерпретации сетей Петри является чисто логическим понятием. Асинхронность событий отражается в предположении о конечности интервалов времени между событиями.

Описание временной соотнесенности двух «точечных» событий выполняется с помощью отношений «раньше», «позже», «одновременно». Отношение «одновременно» в «точечной» интерпретации сетью Петри задать нельзя. По классификации [3] тип параллелизма в этой интерпретации сети Петри называется чередованием (interleaving).

«Точечная» интерпретация не полностью отражает структурные, а следовательно, и поведенческие аспекты сетей. При описании процессов в реальных системах, исследовании их свойств и изучении связей сети можно рассматривать как некоторую синтаксическую форму представления процессов. При «точечной» интерпретации сеть рассматривается как сжатая форма описания последовательности событий.

Рассмотрим пример, взятый из работы [6]. Сеть на рис. 1, *а* порождает две последовательности событий: t_1t_2 и t_2t_1. Эти последовательности составляют все множество возможных строгих упорядочений событий в процессах, порожденных сетью на рис. 1, *а*. Оказывается, что такое же множество последовательностей событий задает сеть с совершенно другой структурой, показанной на рис. 1, *б*. Разница в структурах может быть содержательно истолкована как проявление разных типов внутренней организации процессов, несмотря на одинаковое внешнее поведение.

В другой возможной интерпретации сетей Петри основными понятиями служат операция и переход. Для объектов из мира реальных систем, понимаемых как операции, характерно то, что операции могут выполняться и требуют для выполнения некоторого интервала времени. Таким образом, операция – это «интервальное» событие. В «интервальной» интерпретации предполагается, что сеть описывает процесс выполнения операций, в котором за завершением одной операции сразу же следует запуск следующей. Переход от одной операции к другой происходит мгновенно.

Отсутствие средств описания одновременности отнюдь не означает невозможность описания параллелизма. Понятие «два события происходят параллельно» кроме толкования «одновременно» допускает толкование «между этими событиями не задано (не известно) отношение "раньше-позже"». В системе событий (срабатываний переходов), описываемых сетью Петри, отношение «раньше-позже» задано частично дугами графа, т. е. не на всех парах событий. Пары событий, на которых это отношение не задано, считаются происходящими параллельно. Такими оказываются события, символизируемые переходами, одновременно входящими при текущей маркировке в множество возбужденных переходов. По классификации [3] тип параллелизма в этой интерпретации сети Петри называется истинным.

В «интервальной» интерпретации позиции сети Петри символизируют операции, переходы обозначают окончание одной операции и запуск другой. Динамика изменения маркировки интерпретируется как некоторый поток в сети, который (при работе ЭВМ, например) может быть сопоставлен с потоком управления или данных. Особенно наглядна «интервальная» ин-

терпретация при использовании подкласса сетей Петри, так называемых О-сетей. В О-сетях каждая позиция имеет не более одного входного перехода и не более одного выходного. Наличие маркера в позиции обозначает выполняющуюся операцию. Сеть на рис. 1, *а* принадлежит к классу О-сетей, а на рис. 1, *б* – нет.

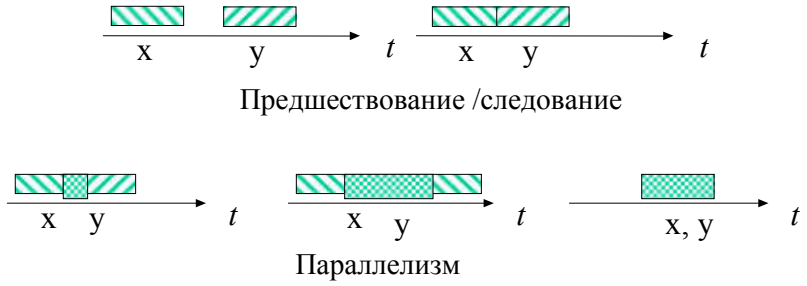


Рис. 2. Возможные отношения между интервальными событиями

Временная соотнесенность интервалов времени может быть задана с помощью бинарных отношений предшествования/следования и параллелизма. Графические изображения этих отношений показаны на рис. 2. Параллелизм между операциями в такой интерпретации понимается именно как одновременность.

Интервальные события могут рассматриваться как пара «точечных», маркирующих начало операции и ее окончание. Несмотря на это, «точечная» и «интервальная» интерпретации взаимно неортогональны. Таким образом, классификация по [3] и этого формализма зависит от его интерпретации.

Используя сети Петри для моделирования реальных систем, можно в зависимости от целей исследования считать временные отношения или заданными предикатами следования и параллелизма или определенными через «точечные» понятия «раньше-позже». Первые удобны при сопоставлении поведения и структуры сетей с реальными системами, вторые – для формулировки и анализа математических свойств сетей.

1.2.2. Описание протоколов сетями Петри. Процесс построения сети Петри для заданного протокола состоит обычно из трех этапов [8]:

– выделения взаимодействующих компонент, т. е. протокольных объектов и среды взаимодействия;

– построения сети для каждого протокольного объекта и среды;

– объединения компонент в единую сеть Петри.

Описание протокольного объекта в виде сети начинается с разработки алгоритма функционирования, в котором выделяется совокупность операций, выполняемых объектом. Для представления сетью алгоритма функционирования важно, чтобы явно определялись условия стыковки операций, отношения следования и параллелизма между ними. Каждому стыку между операциями ставится в соответствие переход сети Петри, а условия окончания и запуска операций отражаются структурой позиций и маркировкой. Таким образом, в отличие от описания с помощью конечных автоматов формализм сетей Петри позволяет представить поведение всей системы в целом.

Описание протоколов сетями Петри дает возможность проверить наличие у протокола многих свойств, определяющих корректность, формальными методами. Решение проблемы корректности при использовании формализма сетей Петри возможно, если условия корректности формулируются в виде требований к структуре и поведению сети. Разработанные в настоящее время методы и средства проверки корректности предназначены для анализа сетей по следующим характеристикам [8].

Проблема достижимости. В заданной сети с заданной начальной разметкой требуется ответить на вопрос, достижима ли некоторая разметка. Эта проблема может быть интерпретирована как исследование протокола на возможность запуска некоторого множества операций или достижения определенного состояния.

Проблема живости. Переход сети называется потенциально живым, если существует достижимая маркировка, при которой этот переход может сработать. Сеть Петри определяется как живая, если все ее переходы живые. Анализ сети на свойство живости позволяет выявить избыточность в протоколе, т. е. описание ситуаций, которые невозможны в данной распределенной системе.

Проблема безопасности. Важным условием правильности работы протокола является отсутствие случаев, когда запущенная операция, не закончившись, запускается снова. Сеть, свободная от таких ситуаций, называется безопасной. Формальным критерием безопасности сети является условие ограниченности разметки сети. Безопасной называется такая сеть, у которой ни при каких условиях не может появиться более одного маркера в каждой из позиций.

Проблема сохранения. Если разметка сети Петри моделирует динамику использования некоторых ресурсов, то важно установить неизменность общего количества ресурса при любом поведении системы. Сеть, гарантирующая сохранение общего количества маркеров в любой достижимой маркировке, называется сохраняющей.

В формализме сетей Петри любое свойство протоколов сводится к свойствам живости или безопасности соответствующей сети. Обычно считается, что если протокол представлен живой и безопасной сетью Петри, то получено достаточно полное свидетельство его корректности.

Прежде чем охарактеризовать методы анализа свойств сетей Петри, следует заметить, что сложность алгоритмов решения этих задач экспоненциально зависит от числа вершин в сети. Поэтому при разработке практически применимых методов большое внимание уделяется вопросам снижения алгоритмической сложности. Существующие подходы к анализу свойств сетей Петри можно разбить на два класса: построение графа достижимости и исследование структуры графа сети.

При анализе свойств сети по графу достижимости трудности возникают на этапе построения этого графа, поскольку его вершинами служат все достижимые маркировки. Множество этих маркировок может быть бесконечным. Даже в конечном случае число вершин графа достижимости может быть очень велико, так как оно экспоненциально растет с увеличением числа вершин сети. Таким образом, даже при относительно небольшом числе вершин в сети для хранения графа достижимости могут потребоваться астрономические объемы памяти. В случае возможности представления графа достижимости в памяти ЭВМ его анализ легко автоматизируется. Свойство живости распознается по наличию пути из любой маркировки в начальную, а безопасность устанавливается анализом маркировок. Об успешном применении этого метода для анализа сетей упоминается в работе [8].

Методы, использующие анализ структуры сети Петри, основаны на аппарате матричного исчисления [7] или на редукции сети. Редукция представляет собой упрощение сети, сохраняющее свойства живости и безопасности. Преобразование применяется многократно до тех пор, пока не получится не редуцируемая далее сеть. Если сеть хорошо упрощается, то свойства живости и безопасности нередуцируемого остатка очевидны. Опыт применения редукционных методов показывает большую эффективность этого способа [8].

1.3. Временная логика

Общность формализма сетей Петри не означает, что по одной и той же сети можно узнать о моделируемой системе все что угодно. Унифицированность анализа имеет и свою оборотную сторону. При анализе конкретной системы требуется моделировать ее различными сетями так, чтобы анализируемые свойства системы были выразимы через живость и безопасность моделирующей сети. Таким образом, для анализа производительности протокола должна использоваться сеть, отличная от той, которая применялась для анализа синхронизации, и т. д.

Задачи анализа систем имеют заметную логическую направленность. Поэтому естественно стремление ставить и решать их с применением логических формализмов. Такой подход имеет и определенное методологическое преимущество, состоящее в том, что он позволяет формулировать задачи непосредственно, а не косвенно с помощью подходящей структуры сети, как в формализме сетей Петри. Однако обычные логические формулы классической логики не подходят для выражения и анализа таких свойств динамических систем, как, например, свойство живости,

потому что в таких формулах нельзя ссылаться ни на какое состояние, кроме текущего. Адекватным аппаратом для анализа свойств живости показала себя временная логика, в которой имеются операторы для задания утверждений о будущих состояниях анализируемой сети [9].

Поведение динамической системы может быть отражено на шкале времени в виде последовательности моментов (рис. 3). Если интервалы между событиями одинаковы, то система называется синхронной, иначе – это асинхронная система.

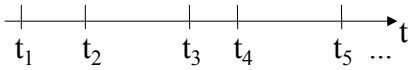


Рис. 3. Цепочка событий

Чтобы непосредственно описать поведение распределенной системы логическими средствами, можно использовать обычное исчисление предикатов и считать время одной из вещественных переменных, от которой зависят все предикаты. Основная идея логического языка временной логики состоит в том, чтобы обойтись без введения в предикаты этой дополнительной переменной. С этой целью используются дополнительные логические операторы, позволяющие выражать временные отношения. Ценой перехода к логическому формализму является невозможность различать синхронные и асинхронные системы. Примерами таких операторов являются:

□A – A справедливо всегда (оператор «всегда»);

◊A – A справедливо иногда (оператор «иногда»).

Формулы временной логики строятся как комбинации предикатов, логических связок и временных операторов и описывают некоторые события, возникающие при функционировании системы. Имеются два способа интерпретации временных формул, различающиеся трактовкой временных операторов.

Можно предполагать, что все события не имеют временной протяженности, т. е. считаются «точечными». В другой интерпретации с событием связывается представление об интервале времени ненулевой длины. В первом случае система в момент события находится в определенном состоянии, а во втором – в течение события проходит ряд состояний. В понятие «всегда» при интервальном понимании событий вкладывается смысл «в каждый момент рассматриваемого интервала». При точечной интерпретации события «всегда» понимается как «в любом состоянии системы в данный момент». Подобным образом различаются трактовки понятия «иногда».

Разница в понимании временных операторов объясняется различием в предположениях о характере времени как логической категории. В «точечном» случае время считается линейным, а исчисление называется логикой линейного времени, в «интервальном» случае – ветвящимся, соответственно исчисление – логикой ветвящегося времени [10]. Эти исчисления взаимно неортогональны. Имеются формулы общезначимые в одной логике и не являющиеся таковыми в другой. Например, формула ⌐□⌐A ↔◊A («не никогда» есть «иногда») общезначима в линейной логике, а в логике ветвящегося времени это утверждение в общем случае неверно [10]. Таким образом, семантика языка временной логики (третьего параметра классификации распределенных систем по [3]) также зависит от интерпретации. Более того, значения этого параметра неортогональны.

Для удобства в исчислении временной логики могут быть использованы временные операторы: ○ (следующий момент) и U (до тех пор, пока), которые можно определить через □ и ◊. В качестве примера рассмотрим, как средствами временной логики описывается простейшее взаимодействие типа: послать/принять (рис. 4). Низкий уровень сигнала *принять* вызовет установку сигнала *послать*. Установка *послать* вызывает установку *принять*. Установка *принять* вызывает сброс *послать*. Сброс *послать* вызывает сброс *принять*. Смысл каждой зависимости может быть описан следующими формулами:

1) □(⌐*принять* → ○*послать*);

2) □(*послать* → *послать* ∨ *принять*);

3) □(*принять* →○⌐*послать*);

4) □(⌐*послать* → ⌐*послать* ∨⌐*принять*).

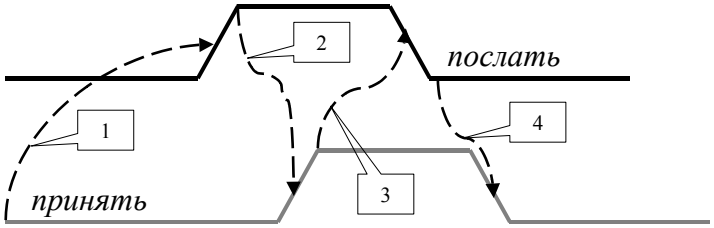


Рис. 4. Временная диаграмма взаимодействия *принять*/*послать*

Формулы временной логики обеспечивают очень естественное задание свойств живости и безопасности. Все свойства безопасности задаются в виде формулы

$$A \to \Box B,$$

где A описывает начальное состояние системы, а B – свойство.

Свойства живости имеют вид

$$A \to \Diamond B$$

с тем же смыслом A и B.

При использовании временной логики проблема корректности протоколов выглядит как задача доказательства непротиворечивости системы логических утверждений, описывающих поведение компонентов и задающих желаемые свойства поведения. Временная логика с успехом использовалась и для реализации протоколов.

Формализм временной логики имеет формы представления утверждений о поведении системы, допускающие *интерпретацию* в виде элементов структуры аппаратуры. Такой формой является представление утверждения в виде множества формул $A \to \circ B$, где формулы A и B не содержат временных операторов. Эта форма задания поведения систем носит название секвенциального автомата. Существует развитый набор методов и средств автоматизации синтеза дискретных устройств, поведение которых задано секвенциальными автоматами [11].

Общность логической формы описания объясняется элементарностью основных понятий. Оборотная сторона этих достоинств состоит в том, что при практическом использовании логические описания поведения протоколов получаются очень громоздкими. Значительный объем описания простых устройств объясняется относительно низким уровнем понятий временной логики и необходимостью вследствие этого кодировки понятий, в которых обычно представлено поведение. С другой стороны, временная логика позволяет компактно формулировать *свойства* поведения анализируемой системы (например, свойства живости и безопасности). Эта особенность широко используется в методе верификации спецификаций распределенных систем, называемом Model Checking [12].

2. Язык ПРАЛУ как средство описания протоколов

Поведение протокола представляет собой описание причинно-следственных и временных зависимостей, позволяющих установить возможные последовательности возникновения событий при функционировании протокола. Анализ поведения дает возможность выявить, реализуются ли при выполнении протокола желательные события, и обнаружить ошибки, когда происходят опасные события. Фундаментальная значимость этих зависимостей проявляется в том, что корректность протокола полностью определяется ими. Язык ПРАЛУ [11, 13] представляет собой попытку использовать более практичные понятия, чем в других формализмах, при сохранении высокого уровня абстракции, характерного для логической теории. Использование ПРАЛУ дает возможность описать временную упорядоченность событий, возникающих при реализации протокола, абстрагируясь от всех деталей, кроме тех, которые выражаются причинно-следственными и временными отношениями.

Для языка ПРАЛУ имеются программные инструменты, выполняющие имитационное моделирование системы по описанию ее поведения. Имитационное моделирование позволяет объединить высокое быстродействие ЭВМ с гибкостью человеческого мышления. Придумывая

эксперимент и интерпретируя его результаты, проектировщик может контролировать неформальные проектные операции. Использование ПРАЛУ для моделирования поведения протоколов позволяет получить информацию о логической корректности непосредственно, а не по результатам оценки статистики сеансов имитационного моделирования. При моделировании с целью анализа логической корректности важно рассматривать таймауты (интервалы ожидания событий), т. е. учитывать параллелизм и асинхронность выполнения операций. Без учета таймаутов опасность ошибочных ситуаций значительно увеличивается.

Язык ПРАЛУ первоначально разрабатывался как средство описания поведения управляющих устройств промышленных агрегатов. Методические материалы по его применению [11, 13] ориентированы в основном на этот класс дискретных устройств. При применении ПРАЛУ для описания протоколов возникает необходимость интерпретации основных понятий ПРАЛУ в терминах поведения протоколов.

2.1. Отображение взаимодействия параллельных процессов

Описание поведения вычислительной сети на языке ПРАЛУ основано на следующем постулате о наблюдаемом поведении компонентов. Информация о внутреннем поведении компонента недоступна для ее внешнего окружения в сети, и наоборот. Отсюда следует, что при описании поведения компонента нужно классифицировать события на протекающие вне компонента и внутри его и описывать их по-разному. Внешние события описываются условиями, накладываемыми на значения переменных, характеризующих внешние информационные связи компонента. Внутренние события в языке ПРАЛУ – это выполнения операций. Предполагается, что каждая операция представляет собой протяженное во времени событие и характеризуется некоторыми моментами ее начала и конца. При функционировании системы происходящие внутри компонента выполнения операций связаны с возникновением определенных внешних событий. Если окончание некоторой операции всегда является следствием некоторого внешнего события, то такая операция называется ожиданием события. Если же наоборот – причиной внешнего события служит окончание операции, то такая операция называется действием.

Описание поведения компонента задается на языке ПРАЛУ алгоритмами, составленными из операций ожидания и действия. Алгоритмы на ПРАЛУ – это неупорядоченные множества цепочек операций. Цепочки представляют собой линейные алгоритмы, в которых сразу после выполнения одной операции запускается следующая. Цепочки снабжены набором меток и заканчиваются операцией перехода. Процесс выполнения алгоритма задается с помощью состояний управления. Цепочка запускается в данном состоянии управления, если все ее метки присутствуют в этом состоянии. После запуска цепочки состояние управления изменяется путем исключения из него меток запущенной цепочки. Цепочки запускаются независимо друг от друга и немедленно после возникновения подходящего состояния управления. Выполнение операций перехода, замыкающих цепочки, состоит во включении в состояние управления перечня меток, указанного в операции. Вследствие этого в алгоритме может быть задано как последовательное, так и параллельное выполнение операций.

Внешние события происходят в точках взаимодействия компонентов. При функционировании компонентов их взаимодействие считается происходящим исключительно через эти точки. Таким образом, точки взаимодействия выделяют интерфейс компонента. Связь (передача сообщений) через точки взаимодействия происходит тогда, когда все компоненты, взаимодействующие через некоторую точку, готовы для взаимодействия. При использовании ПРАЛУ для описания протоколов предполагается, что внешние события отражают факт наличия готовности к взаимодействию, само сообщение не детализируется и информация о значениях данных, их форматах и т. д. не указывается. В этом случае для описания точки взаимодействия достаточно булевой переменной.

События, связанные с завершением операций, состоят в одновременной готовности нескольких точек взаимодействия и могут быть представлены конъюнкциями булевых переменных, сопоставленных этим точкам. Готовность к взаимодействию считается протяженным событием. Внешние события описывают синхронизацию событий (т. е. причинно-следственные и временные зависимости), а не саму передачу сообщений.

Алгоритм на ПРАЛУ задает систему зависимостей между внешними событиями, выражающими их временную соотнесенность. Алгоритм описывает зависимости между каждой парой внешних событий. Эта зависимость может быть или следованием или параллелизмом. Причинно-следственные связи в алгоритмах на ПРАЛУ выражаются парными операциями ожидания и действия, имеющими общую переменную. Причины готовности к взаимодействию выражаются операцией ожидания, а последствия взаимодействия – операцией действия.

Кроме операций, связанных с внешними событиями, в языке ПРАЛУ есть средства представления таймеров и счетчиков. Демонстрационным примером применения языка ПРАЛУ для описания протоколов может служить следующая распределенная система, описанная в [14].

2.2. Пример описания протокола

Система, рассматриваемая в [14], предназначена для организации управления в коммуникационной сети. Систему составляет набор одинаковых функциональных блоков, выполняющих некоторые задания и связанных друг с другом посредством шины, через которую происходит всякое взаимодействие. Блоки организованы в кольцевую структуру.

Протокол управления каналом взаимодействия организует связь между блоками таким образом, что в каждый данный момент времени только один блок управляет шиной (этот блок называется главным). Все другие блоки могут только отвечать на запросы главного блока. Механизм доступа к шине обеспечивает передачу управления доступом по очереди каждому блоку кольца.

Во всех блоках имеется внутренний регистр, который содержит идентификатор блока (его порядковый номер в кольце), считающегося в данный момент главным. Предполагается, что в начальный момент один из блоков является главным, и его идентификатор записан в регистры всех блоков системы. Блок, считающийся в данный момент главным (скажем, блок i), может находиться в этом состоянии фиксированный интервал времени. Когда задание, выполняемое главным блоком, закончится или истечет интервал времени, отведенный для нахождения в состоянии главного блока, главный блок i посылает всем остальным блокам командное сообщение «Стать главным блоку $i+1$» и обновляет содержимое своего собственного регистра на идентификатор $i+1$. После принятия этого сообщения все другие блоки записывают в свои регистры идентификатор $i+1$. Когда блок $i+1$ сделает это, он становится новым главным блоком.

Поведение, описываемое выше, соответствует нормальному случаю, когда при работе системы не происходит ошибок. Предположения о поведении среды передачи и блоков системы при ошибках состоят в следующем:

– сообщения могут теряться как во время передачи, так и во время приема;

– блок может временами становиться «глухим» так, что он теряет любое сообщение, посланное ему;

– блок может временами становиться «немым» так, что любое им посланное сообщение теряется;

– ошибочно работает в каждый данный момент не более одного блока.

Парирование ошибок осуществляется с помощью «будильников» (таймаутов) в каждом блоке и отправки командных сообщений. В каждом блоке по принятии сообщения «Стать главным блоку $i+1$» устанавливается таймаут. Значение этого таймаута выбирается несколько больше допустимой длительности нахождения блока в состоянии «главный блок» с тем, чтобы компенсировать задержку распространения сообщения.

Предположим, что блок i сейчас главный. После истечения своих таймаутов все блоки, за исключением блока $i+2$, увеличивают на единицу содержимое своих внутренних регистров и вновь устанавливают таймауты. Кроме того, блок $i+2$ посылает командное сообщение блоку $i+1$ «Стать главным блоку $i+1$ от $i+2$». Это сообщение удерживается некоторое время, достаточное для восприятия его блоком $i+1$.

Блок $i+1$ считает это командное сообщение недействительным, если его таймаут не истек или идентификатор, записанный в его собственном регистре, отличается от $i+1$. После принятия этого сообщения блок $i+1$ становится главным. Время удержания этого сообщения выбрано таким, что компенсируются небольшие различия между таймаутами в разных блоках.

При наличии ошибок протокол должен гарантировать следующие свойства системы:

– в любой момент не более одного блока находится в состоянии «главный блок»;

– потеря этого состояния в пределах всей системы происходит лишь временно;

– все нормально работающие блоки должны получать состояние «главный блок» в порядке, указанном кольцевой структурой.

Описание поведения одного блока системы на языке ПРАЛУ приведено на рис. 5. Алгоритм, задающий протокол, разбит на подалгоритмы с тем, чтобы структура формального представления соотносилась со структурой словесного описания. Внутренний регистр блока представлен в алгоритме счетчиком *РЕГ*.

```
% Протокол_доступа_к_каналу_блока_СВОЙ
1: >2.3
2: -(РЕГ=СВОЙ)  >*3 >Главный >Инкремент >2.3
  -^ >Прием >*3 >2.3
3: -ТАЙМАУТ >*2 >Восстановление >2.3
% Главный
1: >Задание >Главный_СВОЙ -ВРЕМЯ_УДЕРЖАНИЯ >^Главный_СВОЙ >.
% Прием
1: -Главный_1   >(1=РЕГ)  -^Главный_1 >2
   -Главный_2   >(2=РЕГ)  -^Главный_2 >2
    ....
   -Главный_MAX   >(MAX=РЕГ)  -^Главный_MAX >2
2: >.
% Инкремент
1: -(РЕГ=MAX)  >(РЕГ=0)  >2
  -^ >2
2: >(РЕГ+)  >.
% Восстановление
1: -(РЕГ=СВОЙ-1)  >2.4
  -(РЕГ=СВОЙ+1)  >3.5
  -^ >Инкремент >.
3: >Главный_СВОЙ-1_от_СВОЙ -ВРЕМЯ_УДЕРЖАНИЯ\
     >^Главный_СВОЙ-1_от_СВОЙ
5: >Инкремент >.
2: >Прием >*4 >.
4: -Главный_СВОЙ_от_СВОЙ+1 >*2 >(РЕГ=СВОЙ)  >.
```

Рис. 5. Текст протокола на языке ПРАЛУ

Параметры *СВОЙ, СВОЙ-1, СВОЙ+1* обеспечивают настройку текста на описание поведения конкретного блока. Число блоков в системе задается параметром *MAX*. На месте параметра *СВОЙ* при настройке текста должен стоять идентификатор описываемого блока, а параметры *СВОЙ-1* и *СВОЙ+1* обозначают идентификаторы соседей по кольцу слева и справа. Настройка подалгоритма *ПРИЕМ* зависит от числа блоков, имеющихся в системе. Параметры *ТАЙМАУТ* и *ВРЕМЯ_УДЕРЖАНИЯ* задают конкретные значения параметра таймеров в операциях ожидания времени.

2.3. Корректность алгоритмов

Для языка ПРАЛУ разработана теория корректности алгоритмов, содержащая ряд методов формальной верификации [13, 15]. Наиболее существенными свойствами «правильных» алгоритмов, наличие которых можно проверить формально, являются безызбыточность, восстанавливаемость, непротиворечивость, устойчивость и самосогласованность [13, 7].

Алгоритм безызбыточен, если в нем нет операций, которые никогда не могут быть выполнены. Алгоритм восстанавливаем, если из любого достижимого состояния он может вернуться в исходное. Алгоритм непротиворечив, если при его выполнении не может возникнуть ситуация, в которой одновременно выполняются две операции действия, требующие установки некоторой переменной в противоположные значения. Алгоритм устойчив, если при одновременном выполнении двух операций завершение одной из них не разрушает условий завершения другой. Алгоритм самосогласован, если никакая его операция не может быть повторно запущена, если она не завершилась.

Наличие свойств корректности у системы из нескольких параллельно выполняющихся алгоритмов, полученных настройкой текста на рис. 5, легко установить вручную при неформальном анализе поведения приведенного алгоритма. Таким образом, на рис. 5 показан корректный протокол. Корректность может быть также подтверждена моделированием или автоматической системой доказательства корректности [15].

2.4. Интерпретации ПРАЛУ

В работе [15] показано, что алгоритм на ПРАЛУ можно интерпретировать как сеть Петри. Там же семантика (каноническая интерпретация) ПРАЛУ задается параллельным автоматом, который, в свою очередь, может быть преобразован в форму секвенциального автомата. Секвенциальный автомат можно рассматривать как формулу временной логики. Таким образом, алгоритм ПРАЛУ может интерпретироваться как формула временной логики.

В теории последовательных систем установлено, что формализмы описания поведения эквивалентны по выразительной силе. Аналогичный вывод можно сделать для моделей распределенных систем, рассмотренных в [3]. При практическом применении языка желательно теоретически обосновать выбор модели. Так как рассматриваемые языки равномощны по классу описываемых систем, то можно использовать в качестве критерия богатство возможных интерпретаций формализма. По этому критерию язык ПРАЛУ можно считать более предпочтительным по сравнению с остальными.

Заключение

В работе [1] рассматривались формализмы для описания и анализа протоколов и методы использования этих формализмов. Данная статья является продолжением работы [1]. В ней анализируются ключевые формализмы теории распределенных систем: обобщенные автоматы, сети Петри, временная логика и язык ПРАЛУ. Основная причина использования формальных методов описания распределенных систем состоит в том, что формализмы дают инструментальные средства для анализа моделей протоколов, позволяющие находить ошибки в протоколах и свойства поведения как на уровне спецификации, так и на уровне выполнения. Формальные методы могут использоваться для оценки производительности реализации протоколов и применяться при разработке инструментов синтеза. Формальные методы незаменимы при проверке эквивалентности описаний с различными уровнями абстракции. В настоящее время существуют компьютерные инструментальные средства, свободно распространяемые и доступные за плату, применяющие формальные языки описания протоколов.

Наиболее трудная часть использования формализма, возможно, состоит в выборе языка, соответствующего уровню абстракции модели. При выборе простой модели важные подробности поведения могут оказаться скрытыми, а применение сложной модели может вызывать комбинаторный взрыв в пространстве состояний. В результате анализа известных формальных моделей показывается перспективность языка ПРАЛУ при синтезе реализации протокола и при проверке его корректности. Продемонстрирована его адекватность задаче спецификации причинно-следственных и временных отношений между событиями, а проблема анализа этих отношений играет ключевую роль при анализе логической корректности протоколов и их реализаций.

Имеющиеся программные средства на основе использования ПРАЛУ позволяют моделировать систему протокольных объектов, взаимодействующих через «пустую» среду [16], создавать программы, реализующие протоколы на микропроцессоре К580 [17], синтезировать контроллеры доступа в базисах ПЛМ и ИС малой степени интеграции [18].

Список литературы

1. Черемисинов Д.И. Формальные методы анализа поведения распределенных систем // Информатика. – 2004. – № 3. – С. 17–28.

2. Букур И., Деляну А. Введение в теорию категорий и функторов. – М.: Мир, 1972. – 260 с.

3. Sassone V., Nielsen M., Winskel G. A classification of models for concurrency // 4th International conference on concurrency theory. – Hildesheim, Germany, 1993. – P. 82–96.

4. Börger E. High level system design and analysis using abstract state machines // Current trends in applied formal methods (FM–Trends 98). Lecture notes in computer science. – Vol. 1641. – Springer Verlag, 1999. – P. 1–43.

5. Зандере Л.Я., Мартинсонс К.Я., Фрицнович Г.Ф. Автоматизация разработки протоколов сетей ЭВМ и их реализаций (библиография с комментариями) // Вычислительные сети: логическое проектирование протоколов. – Рига: Зинатне, 1988. – Вып. 2. – С. 200–273.

6. Котов В.Е. Сети Петри. – М.: Наука, 1984.

7. Питерсон Дж. Теория сетей Петри и моделирование систем. – М.: Мир, 1984.

8. Бандман О.Л. Сети Петри и корректность протоколов передачи данных // Вычислительные системы: Математическое и архитектурное обеспечение параллельных вычислений. – Новосибирск, 1985. – Вып. 109. – С. 29–51.

9. Bochman G.V. Hardware specification with temporal logic: An example // IEEE trans. on comp. – 1982. – Vol. 31. – № 73. – P. 222–231.

10. Emerson E.A., Halpern J.Y. «Sometimes» and «Not never» revisited: On branching versus linear temporal logic // JACM. – 1986. – Vol. 33. – № 1. – P. 151–178.

11. Закревский А.Д., Василенок В.К. Описание алгоритмов логического управления. – Мн.: Ин-т техн. кибернетики АН БССР, 1985. – 68 с.

12. Clarke E.M., Emerson E.A., Systla A.P. Automatic verification of finite-state concurrent systems using temporal-logic specifications // ACM Transactions on Programming Languages and Systems. – 1986. – Vol. 8. – № 2. – P. 244–263.

13. Закревский А.Д. Параллельные алгоритмы логического управления. – Мн.: Ин-т техн. кибернетики НАН Беларуси, 1999. – 202 с.

14. Menasche M., Berthomieu B. Time Petri nets for analyzing and verifying time dependent communication protocols // Protocol specification, testing, and verification. Vol. 3. – North-Holland: Elsevier Science, 1983. – P. 161–172.

15. Zakrevskii A.D. The analysis of concurrent logic control algorithms // Lecture Notes in Computer Science, Fundamentals in Computation Theory. – Springer Verlag. – Vol. 278. – 1987. – P. 497–500.

16. Черемисинов Д.И. Визуализация поведения алгоритмов управления. – Мн., 1988. – 28 с. (Препринт / Ин-т техн. кибернетики АН БССР; № 27).

17. Черемисинов Д.И. Кросс-транслятор ПРАЛУ для контроллеров на базе микропроцессора К580. – Мн., 1987. – 46 с. (Препринт / Ин-т техн. кибернетики АН БССР; № 13).

18. Система ЛОГИКА-М синтеза управляющих устройств в базисе ПЛМ и микропроцессоров /А.Д. Закревский, П.Н. Бибило, В.К. Василенок и др. // УСиМ. – 1987. – № 3. – С. 31–35.

Поступила 21.06.04

Объединенный институт проблем
информатики НАН Беларуси,
Минск, Сурганова, 6
e-mail:cher@newman.bas-net.by

D.I. Cheremisinov

THE MORPHISMS OF REACTIVE SYSTEM BEHAVIORAL MODELS

The general idea of formalism usage to describe and reason about protocols is described. The analyzed formalisms are Petri Nets, process algebras, temporal logic and PRALU language. The main reason to use the formal description techniques, or formal methods, is that they give the tools to analyze the protocol models and other systems of interest, and to find the facts and errors in them, both at the specification level and at the implementation level. Formal methods can be used in performance evaluation, and even to find the new implementation techniques. Checking the conformance between various levels of abstraction is also possible. There currently exist many computer tools, both free and commercial, to help us in using formalisms. It is mostly difficult to choose the proper formalisms and the abstraction level of the model. A simple model will possibly leave some important details hidden, whereas a complex model can cause combinatorial explosion in the state space. The result of the analysis of the known formal models is the usability of PRALU language in the protocol design.