УДК 006.72 **DOI** 10.15622/sp.2020.19.3.1

И.В. Афанасьева, Ф.А. Новиков, Л.Н. Федорченко

МЕТОДИКА ПОСТРОЕНИЯ СОБЫТИЙНО-УПРАВЛЯЕМЫХ ПРОГРАММНЫХ СИСТЕМ С ИСПОЛЬЗОВАНИЕМ ЯЗЫКА СПЕЦИФИКАЦИИ CIAO

Афанасьева И.В., Новиков Ф.А., Федорченко Л.Н. **Методика построения событийно-управляемых программных систем с использованием языка спецификации CIAO.**

Аннотация. Событийно-управляемые программные системы в научной литературе относят к классу систем со сложным поведением, называемых реагирующими системами (reactive systems), то есть систем, которые на одно и то же входное воздействие реагируют по-разному в зависимости от своего состояния и предыстории. Такие системы удобно описывать с помощью автоматных моделей с использованием специальных языковых средств – как графических, так и текстовых. Представлена методика автоматизированного построения систем со сложным поведением с использованием разработанного авторами языка CIAO (Cooperative Interaction of Automata Objects), который позволяет на основе неформального описания реагирующей системы формально специфицировать требуемое поведение. Описание реагирующей системы может быть задано словесно на естественном языке или иным способом, принятым в конкретной предметной области. Далее по этой спецификации на языке CIAO специальным преобразователем генерируется программная система взаимодействующих автоматов на языке программирования C++. Сгенерированная программа реализует поведение, гарантированно соответствующее заданной спецификации и исходному неформальному описанию. Для языка CIAO предусмотрена как графическая, так и текстовая нотация. Графическая нотация основана на расширенной нотации диаграмм автомата и диаграмм компонентов унифицированного языка моделирования UML, которые хорошо зарекомендовали себя в описании поведения управляемых событиями систем. Текстовый синтаксис языка CIAO описан контекстно-свободной грамматикой в регулярной форме. Автоматически генерируемый код на языке C++ допускает использование как библиотечных, так и любых внешних функций, написанных вручную. При этом доказательное соответствие формальной спецификации и сгенерированного кода сохраняется при условии соответствия внешних функций своим спецификациям. В качестве примера предложено оригинальное решение задачи Д. Кнута о реагирующей системе управления лифтом. Продемонстрирована действенность предлагаемой методики, поскольку сам автомат-преобразователь, генерирующий код на C++, представлен как реагирующая система, специфицирован на языке CIAO и реализован методом раскрутки. Проведено сравнение предлагаемой методики с другими известными формальными методами описания систем со сложным поведением.

Ключевые слова: модель поведения, системы со сложным поведением, реагирующие системы, граф переходов состояний, синтаксическая граф-схема, грамматика в регулярной форме, шаблоны генерации кода C++.

1. Введение. Основная трудность при проектировании и реализации программных и программно-аппаратных прикладных систем со сложным поведением состоит в непростой и утомительной проверке соответствия требованиям, поскольку исчерпывающее тестирование, как общепринятый метод, для таких систем в принципе

невозможно. Для сложных систем надежность обеспечивается не столько проверкой свойств готового изделия, сколько применением методов, обеспечивающих качество процесса построения систем. Правильно сконструированная система и работает правильно.

Описывается методика разработки асинхронных реагирующих систем [1], состоящая из следующих основных шагов.

1. Исходя из неформального описания задачи, составляется точная формальная спецификация системы в наглядной графической форме путем выделения автоматных объектов и спецификации их поведения в виде набора графов переходов состояний. Спецификации автоматных объектов анализируются и верифицируются вручную, насколько это возможно.

2. По графической спецификации автоматных объектов составляется функционально эквивалентный текст программы на языке CIAO. Этот процесс в настоящее время автоматизирован частично, но в будущем может быть автоматизирован полностью.

3. По описанию системы на языке CIAO автоматически генерируется код программы на языке C++, который гарантированно функционально эквивалентен спецификации на языке CIAO. Генерируемый код может содержать вызовы функций из внешних библиотек, которые разработаны традиционными методами.

Предлагаемая методика не является универсальной – в некоторых случаях ее применение дает положительный результат, в других случаях методика может оказаться практически бесполезной. Прежде чем излагать детали, необходимо охарактеризовать в общем виде тот класс систем, при разработке которых методика дает наилучший результат, как было проверено на практике [2, 3].

Программные системы можно классифицировать на *трансформационные системы* (transformational systems), которые преобразуют входные данные в выходные, и *реагирующие системы* (reactive systems), которые реагируют на изменения внешней среды [1]. Трансформационные системы запускаются по мере необходимости, тогда как реагирующие системы часто являются постоянно действующими. Для описания поведения трансформационных систем достаточно описать связь между начальным и конечным вычислительными состояниями, то есть описать связь между начальными и конечными значениями всех переменных. Порядок действий в этих системах не имеет особого значения, если результат вычислен правильно. В реагирующих системах, напротив, необходимо, чтобы каждому входному событию соответствовало свое определенное действие, и эти действия должны выполняться строго в том порядке, в котором происходят входные события.

При реализации поведения реагирующих систем в большинстве случаев используют одну из двух схем, представленных на рисунке 1.

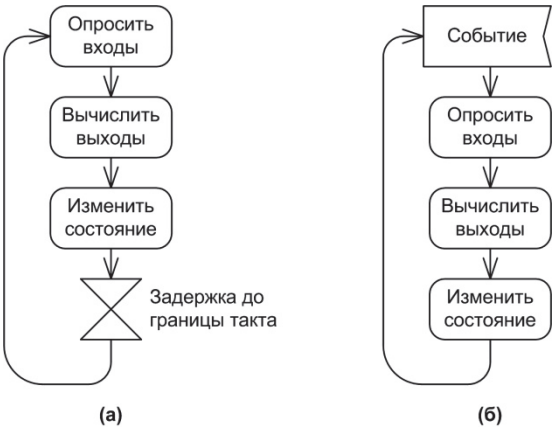

Рис. 1. Реагирующие системы: (а) – с мониторингом; (б) – с прерываниями

Схему (а) называют *синхронной*, а схему (б) – *асинхронной*. Синхронная схема реализации предполагает, что система повторяет цикл действий через фиксированный промежуток времени (такт). Эта схема подходит для моделирования электронных цифровых систем, где выполнение синхронизируется с тактовыми сигналами, а внешние изменения могут происходить между любыми двумя тактами. Асинхронная схема более гибкая и позволяет выполнять несколько циклов в течение одного промежутка времени. Однако обработка прерываний требует дополнительных накладных расходов на организацию очередей событий, переключение контекстов и так далее. За последнее время вычислительная мощность компьютеров значительно выросла, поэтому сейчас реагирующие системы чаще оказываются асинхронными, нежели синхронными.

В настоящее время известны многие методы разработки реагирующих систем, которые базируются на моделях поведения с явным выделением состояний. Необходимо отметить метод Statemate и диаграммы состояний Харела [1, 4], методологию разработки систем реального времени ROOM [5], метод архитектурного проектирования и моделирования параллельных объектов COMET [6, 7] и автоматное программирование (SWITCH-технологию) [8].

Все указанные источники были использованы при разработке предлагаемой методики. Методика имеет достаточно богатую предысторию развития и обширные основания. Опираясь на имеющийся практический опыт разработки систем управления [2] и

используя опробованные ранее методы, такие как конструирование специализированных языков предметной области [9], применение синтаксически управляемой обработки данных [10], использование графических средств моделирования предметных областей [11], предлагается методика, продолжающая и развивающая как предшествующие разработки авторов данной статьи, так и разработки коллег. Отличие предлагаемой методики от предшественников заключается в изменении целеполагания. Известные методы нацелены на разработку «больших» и «тяжелых» промышленных систем с неограниченной областью применения. Главная цель при этом – сделать в срок. Предлагаемая методика хорошо подходит для разработки небольших и малых систем с четко очерченной областью применения. Главная цель – сократить удельные трудозатраты на разработку при сохранении приемлемого качества продукта. Эта цель при применении методики достигается, что подтверждается ее практическим применением [12].

Статья имеет следующую структуру. В разделе 2 рассматриваются различные средства спецификации поведения реагирующих систем и позиционируются теоретические основы предлагаемой методики. Раздел 3 представляет графическую форму языка спецификации. В разделе 4 формулируется задача, выбранная в качестве сквозного примера для демонстрации методики. Речь идет о реагирующей системе управления лифтом, которую Дональд Кнут разбирает в первом томе своей знаменитой монографии [13]. Раздел 5 описывает методику перевода спецификации CIAO в программу на C++. В разделе 6 обсуждается получение текстовой спецификации CIAO из формальной визуальной модели. Раздел 7 стоит несколько особняком: фактически в нем приводится еще один пример применимости предлагаемой методики, а именно показано, как язык CIAO реализуется средствами языка CIAO, то есть проводится раскрутка. В разделе 8 возвращаемся к основному примеру и показываем, как синтезировать целевую программу управления лифтом. В заключительном разделе 9 проводится сопоставление предлагаемой методики с другими методами, и в Заключении кратко подводятся итоги и намечаются направления будущих исследований.

Цель исследования – предложить методику для автоматизированной реализации реагирующих программных систем с помощью языка CIAO [2, 3, 12, 14, 15]. В основе языка CIAO лежат диаграммы автомата, подобные диаграммам UML, которые, как показал опыт, могут наглядно представлять поведение управляемых событиями систем.

2. Спецификация поведения реагирующих систем. Современные реагирующие системы являются комплексами в том смысле, что выполняют множество функций, управляют многими объектами (здесь объекты – это объекты управления в самом обычном смысле) и зачастую распределены на несколько вычислительных устройств. Реализация реагирующей системы в форме одного монолитного алгоритма не отвечает современным требованиям, поэтому применение объектно-ориентированной парадигмы имеет очевидные конкурентные преимущества и является общепринятым.

Таким образом, реагирующая система представляется как система кооперативно взаимодействующих программных объектов. Каждый такой программный объект отвечает за какой-то один аспект поведения или функцию всей системы, и его поведение может быть описано хорошо известным механизмом описания поведения – графом переходов состояний, или как принято говорить – *автоматом*. Поскольку каждый отдельный автомат встроен в свой программный объект, мы называем его *автоматным объектом*. Поведение системы в целом складывается из поведения автоматных объектов, взаимодействующих через определенные интерфейсы. Автоматные объекты могут и должны выполняться на различных устройствах, поэтому сразу рассматривается случай гетерогенных систем. В этом заключается основная идея и мотивация языка CIAO – описать возможные сценарии поведения сложных гетерогенных и распределенных реагирующих систем.

Основными элементами языка CIAO являются автоматные объекты, содержащие графы переходов состояний, и интерфейсы различных типов, как показано на рисунке 2.

Рис. 2. Автоматный объект: (1) и (2) – предоставляемые интерфейсы, (3) и (4) –требуемые интерфейсы

В соответствии с принципом программирования Command-Query Separation (CQS) Б. Мейера [16, 17] операции интерфейса разделяются на две категории: запросы (*query*) – не меняющие состояния и доставляющие значения за пределы объекта, и команды (*command*) – меняющие состояние объекта, но не доставляющие значения наружу.

С учетом типов и категорий интерфейсов всего возможны четыре комбинации:

– предоставляемая команда – это *событие*, на которое автоматный объект готов реагировать;

– требуемая команда – это *эффект* или *действие*, которое автоматный объект требует выполнить;

– предоставляемый запрос – это *текущее состояние* (в том числе значения локальных переменных), которое автоматный объект сообщает внешним объектам;

– требуемый запрос – это *сторожевое условие*, которое проверяется на переходах автоматного объекта.

Таким образом, концепция автоматного объекта в языке CIAO полностью соответствует объектно-ориентированной парадигме: предоставляемая команда соответствует декларации метода, требуемая команда соответствует вызову метода изменения данных объекта, предоставляемый запрос соответствует декларации открытых свойств; требуемый запрос соответствует получению открытых свойств объекта. Отсюда ясно, что концепция автоматного объекта наилучшим образом обеспечивает потребности объектно-ориентированного описания поведения гетерогенных реагирующих систем.

3. Графическая форма языка спецификации CIAO. Прежде чем переходить к изложению подхода к созданию событийно-управляемых программных систем, необходимо уточнить формализмы и языки, положенные в основу методики.

Для спецификации поведения реагирующих систем предлагается самостоятельный графический язык спецификации CIAO, построенный на основе унифицированного языка моделирования UML [11, 18] с использованием и модификацией некоторых конструкций диаграммы автомата, диаграммы компонентов и диаграммы классов.

Графический язык CIAO предназначен для:

– спецификации и визуализации сложного поведения;

– публикации (параллельных) алгоритмов;

– построения имитационных моделей сложного поведения и доказательного исследования этих моделей;

– быстрого прототипирования программного обеспечения гетерогенных реагирующих систем.

Следуя традиции, введнной в практику авторами UML [11, 18], семантика графического языка описывается диаграммой классов, представляющей взаимосвязи между основными понятиями языка (такая диаграмма обычно называется метамоделью), а графическая нотация иллюстрируется примерами, если это нужно. На рисунке 3 представлена метамодель языка CIAO.

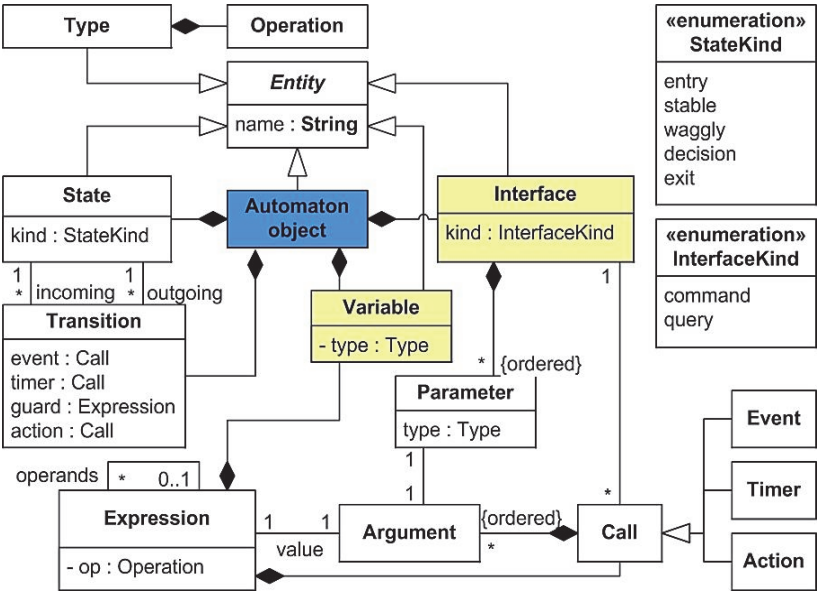

Рис. 3. Метамодель языка CIAO

Для сравнения и указания основных нововведений языка CIAO на рисунке 4 приведена метамодель диаграммы автомата UML.

Основные графические компоненты метамоделей – состояние (*state*) и переход (*transition*), основные неграфические компоненты – событие (*event*), действие (*action*) и сторожевое условие (*guard*).

Метамодель CIAO дополнена стереотипами «интерфейс» (*interface*) и «переменная» (*variable*). Для каждого объекта в модели обязательно указываются интерфейсы, которые он предоставляет, и интерфейсы, которые ему требуются.

В список типов состояний добавлены устойчивые (*stable*) и неустойчивые (*waggly*) состояния. Переход из устойчивого состояния возможен только по событию или по таймеру (*timer*), из неустойчивого – по завершению деятельности в состоянии, без

проверки сторожевых условий. Состояния ветвления и соединения, а также исторические и составные состояния метамодели UML в метамодели CIAO не используются.

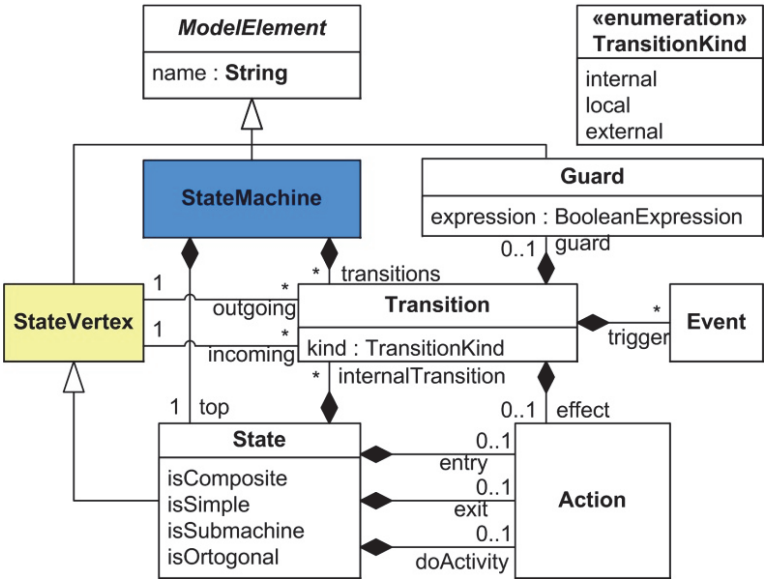

Рис. 4. Метамодель диаграммы автомата UML [18]

Автоматный объект является классификатором в смысле языка UML и потому обозначается прямоугольником; состояния и интерфейсы трактуются и обозначаются, как это принято в языке UML 2. Переходы между состояниями на диаграмме изображаются в виде стрелочек, нагруженных указанием события, сторожевого условия и эффекта перехода (действий) в текстовом виде.

Событие должно быть предоставляемой командой, эффект – одной или более требуемыми командами, либо выражением. Сторожевое условие имеет вид произвольной булевой формулы над требуемыми запросами и значениями локальных переменных. Синтаксис всех этих конструкций принят такой же, как в обычных языках программирования: переменные обозначаются в текстовом виде; выражения, в частности сторожевые условия, составлены из имен переменных и вызовов функций с помощью знаков операций; события и действия оформляются как операторы вызова функций без результата (процедур). Особо оговорим синтаксис предоставляемых запросов. По определению, запрос не меняет

состояния, поэтому запрос – это петля, присущая всем состояниям. Возвращаемое значение запроса явно прописывается в описании интерфейса.

Нотация автоматного объекта в виде диаграммы автомата представлена на рисунке 5.

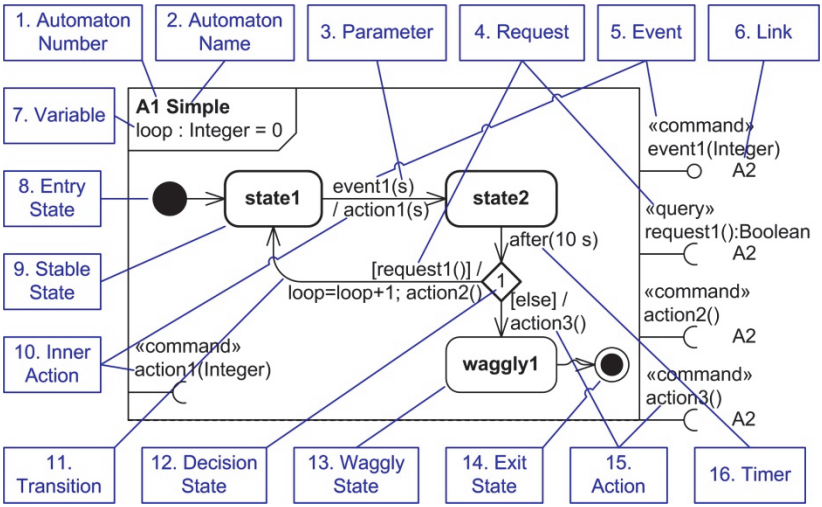

Рис. 5. Графическая нотация автоматного объекта

В каждом автоматном объекте присутствуют следующие элементы:

1. Number, name – порядковый номер и имя автоматного объекта.

2. Variables – набор переменных.

3. States – набор состояний. Устойчивые и неустойчивые состояния идентифицируются уникальными именами, состояния выбора (decision state) – порядковым номером.

4. Transitions – набор переходов между состояниями.

5. Interfaces – набор команд и запросов.

6. Links – связи автоматного объекта с другими компонентами системы.

4. Формулировка задачи о лифте. В своей фундаментальной монографии Д. Кнут разобрал задачу о проектировании системы управления пассажирским лифтом [13]. Система управления лифтом – яркий пример программы, управляемой событиями, и эта задача была выбрана (с незначительными упрощениями ради краткости изложения)

для иллюстрации возможностей спецификации поведения средствами языка CIAO.

Дано пятиэтажное здание с одним пассажирским лифтом. Этажи пронумерованы от 1 до 5. В исходном положении пустой лифт стоит на первом этаже. Требуется разработать программу перемещения лифта между этажами в здании в соответствии со следующими правилами.

1. Каждый этаж имеет одну кнопку для вызова лифта. Система управления лифтом не принимает новых вызовов, пока не закончит обслуживание принятого вызова.

2. В кабине лифта имеется панель с пятью кнопками для перемещения на конкретный этаж. Когда лифт прибывает на соответствующий этаж или когда пассажир не вошел в лифт, панель с кнопками готова принять новый запрос перемещения.

3. В пол лифта встроен датчик веса. Когда пассажир входит в лифт, значение его веса добавляется к суммарному весу пассажиров, уже находящихся в лифте, при выходе из лифта вес пассажира вычитается. Перемещение лифта осуществляется при суммарном весе не более 400 кг. Чтобы лифтом не баловались дети, лифт с весом ниже 30 кг также не поедет.

4. Если нет вызовов на этажах, лифт должен оставаться в конечном пункте назначения с закрытыми дверями и ожидать дальнейших запросов.

5. Когда лифт без пассажира прибывает на соответствующий этаж, в нем включается освещение, и двери открываются. Если пассажир вышел из лифта, через 15 секунд двери закрываются, и освещение лифта гаснет. Если последний пассажир не хочет выходить на этаже прибытия, он может нажать на кнопку другого этажа.

Требуется доказать правильность функционирования системы управления в следующем смысле.

1. Все запросы перемещения на этажи внутри лифта должны в конечном итоге быть обслужены.

2. Все запросы вызова лифта от этажей должны в конечном итоге быть обслужены.

Система формализована с помощью переменных и объектов, перечисленных в таблице 1.

Пассажир лифта (объект Passenger) выполняет следующие действия:

1. Начало работы. Пассажир на этаже inF нажимает кнопку вызова (call(inF)).

2. Ожидание открытия дверей вне лифта (состояние WaitOut). Когда двери открылись (событие opened()), пассажир входит в лифт. Датчик веса увеличит суммарный вес w на вес пассажира wt. Пассажир нажимает кнопку нужного этажа outF.

3. Ожидание открытия дверей внутри лифта (состояние WaitIn). Когда двери открылись (событие opened()):

– Пассажир может захотеть выйти из лифта (exit==true), и выходит, тогда датчик веса уменьшит суммарный вес w на вес пассажира wt. Перейти к шагу 4.

– Иначе пассажир выбирает другой этаж outF, на который он хочет переместиться. Перейти к шагу 3.

4. Конец работы.

Таблица 1. Описание переменных и объектов системы управления лифтом

	Описание	Значение
inF	Номер этажа, с которого пассажир вызвал лифт	1–5
outF	Номер этажа, на который хочет переместиться пассажир	1–5
wt	Вес пассажира (кг)	
exit	Готовность пассажира выйти из лифта, когда откроются двери	true, false
floor	Номер этажа, на котором сейчас находится лифт	1–5
w	Суммарный вес пассажиров лифта (кг)	
door	Двери лифта (могут открываться и закрываться)	
light	Лампочка (может включаться и выключаться)	
cab	Кабина лифта (может перемещаться на заданный этаж)	

Опишем поведение системы управления лифтом (объект Control) по шагам.

1. Начало работы.

2. Ожидание вызова (состояние Idle). Если вызов inF поступил с этажа, на котором сейчас находится лифт, перейти к шагу 4.

3. Переместить лифт на этаж inF.

4. Включить освещение лифта. Открыть двери лифта и двери на этаже.

5. Ожидание запроса на перемещение (состояние Service).

– По прошествии 15 сек, если лифт пустой, закрыть двери лифта и двери на этаже. Выключить освещение лифта. Перейти к шагу 2.

– Если лифт не пустой и нажата кнопка этажа, перейти к шагу 6.

6. Переместить лифт на этаж outF.

– Если лифт уже находится на этаже outF, перейти к шагу 5.

– Если все в порядке (вес в пределах нормы), закрыть двери лифта и двери на этаже. Переместить лифт на этаж outF. Открыть двери лифта и двери на этаже. Перейти к шагу 5.

– Иначе перейти к шагу 5.

Работа механизмов лифта (объект Elevator) описывается очень просто. Когда лифт стоит (состояние Stay), он может включить или выключить освещение (light), открыть или закрыть двери лифта и двери на этаже (door), определить вес вошедших пассажиров (w), принять команду на перемещение кабины лифта (cab) на заданный этаж. Во время перемещения кабины лифт находится в состоянии Move и других команд не принимает.

Приведенное выше словесное описание моделирует работу лифта. Такое описание привычно и кажется понятным, но оно длинное, неточное и может быть противоречивым.

На основании словесных моделей построена система управления лифтом. Работа лифта моделируется с помощью трех компонентов (автоматных объектов), один из которых описывает поведение пассажира (рис. 6), другой – поведение системы управления лифтом (рис. 7), а третий описывает работу механизмов лифта (рис. 8). Эти объекты реализуют все выполняемые действия.

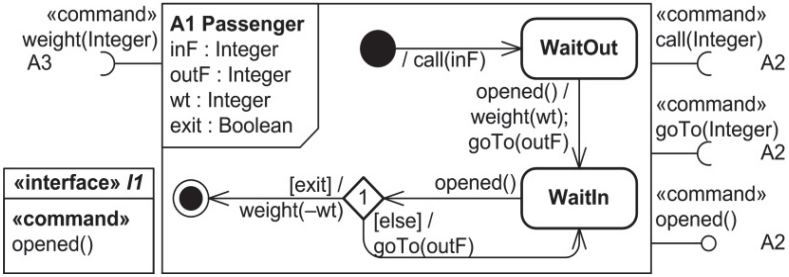

Рис. 6. Автомат A1 (поведение пассажира) и интерфейс I1

Для строгой математической проверки того, что полученная спецификация действительно удовлетворяет требованиям 1 и 2, достаточно проследить все возможные пути в графах переходов состояний и убедиться, что они реализуемы и заканчиваются в требуемых состояниях. Действительно, проверим для примера сначала более простое первое требование – все запросы перемещения на этажи внутри лифта должны быть обслужены. Рассмотрим сценарий: пассажир уже находится в лифте и нажимает

кнопку outF. После того как лифт приедет на нужный этаж, пассажир не выходит из лифта, а выбирает новый этаж. Последовательность событий/действий (то есть путь в графе переходов состояний) в этом случае начинается в состоянии Service автомата A2 и, как легко проверить, завершается в этом же состоянии через конечное число шагов. Последовательность действий для этого сценария представлена в таблице 2. Событие opened() пассажир получает каждый раз, когда дверь лифта открывается или остается открытой после каких-то действий.

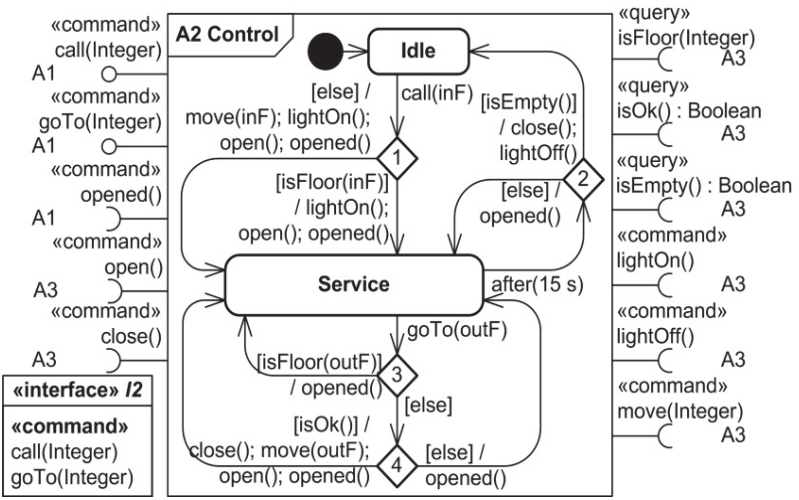

Рис. 7. Автомат A2 (система управления лифтом) и интерфейс I2

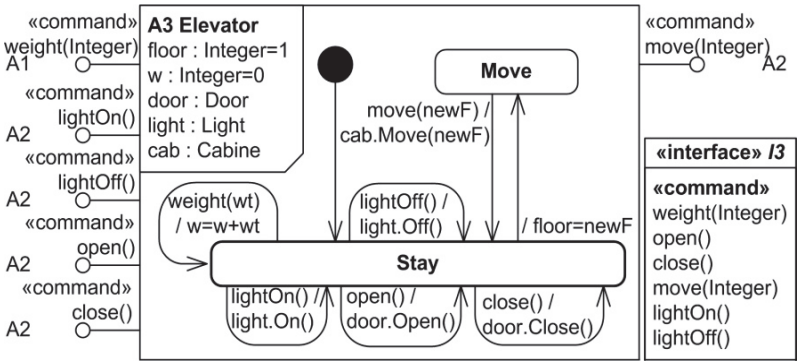

Рис. 8. Автомат A3 (работа механизмов лифта) и интерфейс I3

Таблица 2. Последовательность действий сценария управления лифтом

Автомат	Откуда	Событие	Условие	Действие	Куда
A2	Service	goTo(outF)	!isFloor(outF) & isOK()	close() move() open() opened()	Service
A3	Stay	close()	–	door.close()	Stay
A3	Stay	move(newF)	–	cab.move(newF)	Move
A3	Move	–	–	floor = newF	Stay
A3	Stay	open()	–	door.open()	Stay
A1	WaitIn	opened()	!exit	goTo(outF)	WaitIn

5. Методика перевода спецификации CIAO в программу на языке C++. На примере задачи о системе управления лифтом видим, что получаемые графические спецификации с одной стороны формальны и обладают доказательной силой, а с другой стороны достаточно наглядны и удобны для изучения человеком. Однако целью является получение не только документации к программе в графической форме, но и получение кода самой программы на языке программирования C++.

Рассмотрим полную схему предлагаемой методики в форме диаграммы потоков данных (data flow diagrams) [19], выраженной диаграммой деятельности на языке UML (рис. 9). Прямоугольники на этой схеме означают *данные*, фигуры со скругленными углами означают *процессы* обработки данных, а стрелки показывают входные и выходные потоки данных процессов.

Рис. 9. Схема применения предлагаемой методики

Процессы, выполняемые человеком и связанные с построением спецификации в графической форме (процессы 4 и 6 на диаграмме), освещены в разделе 4. Здесь мы обращаемся к автоматическим процессам перевода полученной графической спецификации в работающую реагирующую систему. Существуют различные варианты такого перевода. Например, можно построить интерпретатор непосредственно диаграмм [9].

Однако в практических промышленных приложениях предпочитают получать автоматически исполняемый код на языке программирования на основе спецификации трансляций. Удобно воспользоваться давно проверенными методами синтаксически управляемой обработки данных, использующих контекстно-свободный язык, который определяется трансляционной КСР-грамматикой (контекстно-свободной грамматикой в регулярной форме) [10].

Принцип синтаксического управления для создания программ подразумевает, что управление процессом обработки данных определяется при помощи синтаксической структуры. В случае трансляции языка программирования управление процессом трансляции задается синтаксической структурой предложений транслируемого языка. Возможен и другой пример применения технологии синтаксического управления – это задание структуры некоторого вычисления. В этом случае управление трансляции инициируется последовательностью состояний вычислительного процесса [20].

Особенность такого подхода состоит в том, что структура входных данных (входного языка) и система управления обработкой данных такой программы определяется сходным образом: одной и той же формальной грамматикой или системой конечных автоматов, или же управляющей таблицей. Главная часть логики такой программы сосредоточена в схеме взаимодействующих автоматов, а сама программа обращается к ней на каждом шаге ее работы. Управляющая таблица автоматов определяет порядок вызова семантических процедур, имена которых записываются в текст детерминированной трансляционной КСР-грамматики, осуществляющих обработку данных. Трансляционная грамматика отличается от классической порождающей грамматики тем, что ее описание содержит дополнительные разделы и вычислительную среду (environment) в виде семантических процедур.

Таким образом, встает задача получения такого текстового представления графической спецификации, которое возможно автоматически преобразовать в код на языке программирования.

6. Представление формальной спецификации в текстовой форме. Основные определения даны в работах [10, 21]. Напомним их неформально.

Как известно, формальным языком является набор цепочек языковых токенов (лексем или терминалов) в соответствии с грамматикой языка, которая состоит из контекстно-свободной части в виде КСР-правил и контекстно-зависимой части (ограничений). КСР-правила представлены в форме с обобщенными регулярными выражениями. Обобщение конечно-автоматной модели обработки языков сводится к следующему:

– в классические регулярные выражения вводится обобщенная итерация, обозначаемая знаком «диез» (#). Обобщенная итерация не расширяет множество регулярных слов и может быть определена через традиционную (одноместную) операцию Клини (*) как (P#Q) = P, (Q, P)*. Она удобна при работе со стеком и частично решает задачу минимизации регулярного выражения по числу вхождений символов из объединенного алфавита всех символов грамматики;

– язык описывается с помощью КСР-грамматики – обобщения КС-грамматики. Класс КСР-языков не расширяет это обобщение, но снимает лишнюю структурированность описания языка, в частности, простые конструкции, например последовательности, списки с разделителями описываются без рекурсивных правил, только с применением итерации;

– строится синтаксическая граф-схема (СГС) – графический аналог КСР-грамматики, стартовый объект для синтеза распознавателя (анализатора) языка.

Грамматическое описание языка CIAO. Правила грамматики записываются следующим образом:

Нетерминал : Регулярное_выражение,

где Нетерминал – одно из вышеперечисленных обозначений для нетерминалов, а Регулярное_выражение задается следующим синтаксисом в формализме Наура-Бэкуса:

```
Регулярное_выражение ::= {Пусто | Лексема |
 Нетерминал | Семантика |                    // Базовые элементы (1)
 Регулярное_выражение1                      // Конкатенация (2)
Регулярное_выражение2 |
 Регулярное_выражение1 ';'                  // Альтернативный выбор (3)
Регулярное_выражение2 |
 Регулярное_выражение1 '#'                  // Итерация (4)
Регулярное_выражение2 |
 '[' Регулярное_выражение ']' |             // Необязательный элемент (5)
 '(' Регулярное_выражение ')' }.            // Выражение в скобках (6)
```

В фигурных скобках через вертикальную черту перечислены альтернативы. В строке (1) перечислены базовые элементы, составляющие регулярное выражение: Пусто – пустое выражение (отсутствие чего-либо), Лексема, Нетерминал и Семантика. Строка (2) представляет операцию конкатенации, не имеющую специального знака для своего обозначения; строка (3) – это операция альтернативного выбора, знаком которой является точка с запятой; строка (4) задает операцию итерации, знаком которой является диез, строка (5) задает необязательную конструкцию, то есть [P] = (P ;), а строка (6) позволяет заключать регулярное выражение в круглые скобки, чтобы рассматривать его как один операнд в операциях конкатенации, альтернативного выбора и итерации. Для обозначения комментария используется комбинация символов //.

Ниже описан конкретный синтаксис языка CIAO с минимумом разделителей. Интенсивно используется итерация, рекурсивные правила в явно выписанной части грамматики не используются.

В текущей версии грамматики языка CIAO выделяются следующие 11 нетерминалов для обозначения отдельных языковых структур: P (описание одного автоматного объекта на языке CIAO), N (номер и имя автоматного объекта), E (раздел событий – входных команд), A (раздел действий – выходных команд), R (раздел запросов), U (раздел устойчивых состояний), W (раздел неустойчивых состояний), D (раздел состояний выбора), L (раздел внешних связей), Stt (общий нетерминал для всех видов состояний), Act (список действий и/или выражений на переходе). Начальным нетерминалом, из которого порождается текст любой синтаксически правильной программы, является нетерминал P. Полная программа на CIAO является последовательностью разделов в фиксированном порядке.

Лексический анализатор преобразует цепочку символов из входного файла в последовательность лексем.

В языке CIAO можно выделить следующие группы лексем:

1) лексемы-названия разделов:

'EVENT', 'ACTION', 'REQUEST', 'STATE', 'WAGGLY', 'DECISION', 'LINK';

2) лексемы общего вида (в их обозначениях присутствуют угловые скобки < и >):

– '<p_num>' – уникальный номер автоматного объекта в пространстве имен, записывается так: #целое;

– '<p_nm>' – имя автоматного объекта, произвольный идентификатор;

– '<tag>' – имя параметра, произвольный идентификатор;

– '<type>' – идентификатор встроенного типа, то есть один из идентификаторов 'string', 'Real', 'Integer', 'Boolean', или идентификатор внешней структуры (класса);

– '<expr>' – логическое или арифметическое выражение;

– '<e_nm>' – имя события – входной команды (часть предоставляемого интерфейса), произвольный идентификатор;

– '<a_nm>' – имя действия – выходной команды (часть требуемого интерфейса), произвольный идентификатор;

– '<r_nm>' – имя запроса, произвольный идентификатор;

– '<u_nm>' – имя устойчивого состояния, произвольный идентификатор;

– '<w_nm>' – имя неустойчивого состояния, произвольный идентификатор;

– '<d_nm> – имя состояния выбора, записывается так: `dcsn`целое. Под целым числом подразумевается номер состояния выбора;

– '<f_num>' – номер вызывающего автоматного объекта, которому данный автомат предоставляет указанный интерфейс, записывается так: #целое;

– '<t_num>' – номер вызываемого автоматного объекта, чей указанный интерфейс требуется, записывается так: #целое;

– '<s_num>' – число секунд для интервала времени: либо натуральное число, либо число с плавающей точкой;

– '<val>' – самоопределенное значение: либо «строка», либо истинностное значение 'T', 'F', либо натуральное число, либо число с плавающей точкой;

3) лексемы-зарезервированные слова: 'Real' – тип число с плавающей точкой; 'Integer' – целочисленный тип; 'Boolean' – булевский тип; 'after' – выход из состояния по прошествии указанного интервала времени; 'entry' – начальное состояние; 'exit' – заключительное состояние; 'T' – значение «истина» для булевского типа; 'F'– значение «ложь» для булевского типа.

Для учета неформализованных контекстных зависимостей в правила грамматики в виде регулярных выражений введены следующие 15 семантик – процедур, которые исполняются, если в процессе распознавания входного текста очередная распознанная лексема является той, которая в грамматическом правиле следует за

данной семантикой; причем эта семантика имеет доступ ко всем параметрам данной лексемы. Название каждой семантики начинается со знака '**$**': `$open`, `$e1`, `$a1`, `$r1`, `$r2`, `$e2`, `$after`, `$s1`, `$entry`, `$if`, `$else`, `$l1`, `$l2`, `$close`, `$a2`.

Для проверки правильности записи регулярных выражений использовалось инструментальное средство SynGT (Syntax Graph Transformations) [10, 21]. Графовая форма представления грамматик оказалась столь же естественной и для постановки на ней задачи автоматической генерации тестов, как и задачи построения управляющих автоматов. Небольшая модификация компонентов граф-схемы, превращающая ее в сеть, позволяет свести задачу генерации оптимального теста к сетевой задаче нахождения максимального потока при минимальной стоимости, которая на сетях всегда имеет решение. Грамматика CIAO в регулярной форме приведена в листинге 1.

```
P : N E [A] [R] U W [D] L .
N : <p_num> <p_nm> $open .
E : EVENT ( <e_nm> ( ( ; <tag> : <type> ) ) $e1 ) # .
A : ACTION ( <a_nm> ( ( ; <tag> : <type> ) ) $a1 ) # .
R : REQUEST ( <r_nm> ( ( ; <tag> : <type> ) ) $r1 : <type> $r2 ) # .
U : STATE ( <u_nm> -> ( <e_nm> ( ( ; <val> ) ) $e2 ;
    after ( <s_num> ) $after ) ( ; Act ) -> Stt $s1 ) # .
W : WAGGLY entry -> ( ; Act ) -> <u_nm> $entry
    ( ; <w_nm> -> ( ; Act ) -> <u_nm> $s1 ) # .
D : DECISION ( <d_nm> -> ( [ <expr> ] $if ( ; Act ) -> Stt $s1
    | $else ( ; Act ) -> Stt $s1 ) ) # .
L : LINK ( <f_num> <p_num> <e_nm> $l1 ;
    ( ; <p_num> <t_num> ( <a_nm> ; <r_nm> ) $l2 ) ) # $close .
Act : / ( <a_nm> ( ( ; <val> ) ) $a2 ) # ( , ) .
Stt : <u_nm> ; <d_nm> ; <w_nm> .
```

Листинг 1. Грамматика CIAO в регулярной форме

Графическим элементам автоматных объектов соответствуют конструкции (шаблоны проектирования) на текстовом языке CIAO [3, 12]. Используя данные шаблоны, по диаграмме автомата вручную строится текст на языке CIAO. Например, листинг 2 соответствует автомату A2 Control, реализующему поведение системы управления лифтом (рис. 7).

Интерфейс I2 представлен в разделе EVENT, логика выполнения алгоритма (состояния и переходы) сосредоточена в разделах STATE, WAGGLY и DECISION. Действия на переходах перечислены в разделе ACTION, запросы (стереотип «*query*») содержатся в разделе REQUEST. В разделе LINK отображены

связи (предоставляемые и требуемые интерфейсы) автомата A2 с автоматами A1 и A3.

```
#2 Control
EVENT
call(y:Integer)
goTo(y:Integer)
ACTION
opened()
lightOn()
lightOff()
open()
close()
move(y:Integer)
REQUEST
isFloor(y:Integer):Boolean
isOk():Boolean
isEmpty():Boolean
STATE
idle -> call(inF) -> dcsn1
service -> after(15 s) -> dcsn2
service -> goTo(outF) -> dcsn3
WAGGLY
entry -> -> idle
DECISION
dcsn1 -> ( [isFloor(inF)] /
  lightOn(), open(), opened()
  -> service | / move(inF),
  lightOn(), open(), opened()
  -> service )
dcsn2 -> ( [isEmpty()] / close
  (), lightOff() -> idle | /
  opened() -> service )
dcsn3 -> ( [isFloor(outF)] /
  opened() -> service | ->
  dcsn4 )
dcsn4 -> ( [isOk()] / close(),
  move(outF), open(), opened()
  -> service | / opened() ->
  service )
LINK
#1 #2 call
#1 #2 goTo
#2 #1 opened
#2 #3 isFloor
#2 #3 isOk
#2 #3 isEmpty
#2 #3 lightOn
#2 #3 lightOff
#2 #3 open
#2 #3 close
#2 #3 move
```

Листинг 2. Текстовая спецификация системы управления лифтом

7. Автомат-преобразователь – раскрутка языка CIAO. Покажем действенность и работоспособность предлагаемой методики на примере самоприменимости, а именно специфицируем на языке CIAO программу, которая переводит спецификации на языке CIAO в программы на языке C++. Тем самым выразительная сила языка CIAO окажется продемонстрированной в полной мере.

В системе используются следующие глобальные переменные (табл. 3).

На вход подается текст на языке CIAO. Требуется транслировать его в программу на языке C++.

Автомат-преобразователь (A1 Translate) реализован на языке CIAO методом раскрутки.

1. Автомат A1 реализует только логику. Он запускается каждый раз для нового текста.

2. Все операции работы с файлами реализует автомат головной программы (A2 Main).

3. Поскольку текст программы является промежуточным, мы вправе считать, что в нем все правильно отформатировано: удалены комментарии (от символов ‘//’ до конца строки) и пустые строки, в одной строке содержится одна лексема-название раздела или одно законченное предложение, допустимое языком CIAO.

4. Все успешно выполненные интерфейсные операции головной программы заканчиваются чтением следующей строки s из файла и вызовом события ok(s).

Таблица 3. Глобальные переменные системы трансляции

	Описание
s	Текст текущей строки программы на языке CIAO
n	Тип строки для распознавания: 1 – содержит номер и имя автомата 2 – содержит событие 3 – содержит действие 4 – содержит запрос 5 – содержит устойчивое состояние 6 – содержит неустойчивое состояние 7 – конец файла (eof)
er	Наличие ошибки (есть или нет)

На основе словесного описания построена система трансляции, состоящая из двух компонентов – автомата-преобразователя A1 Translate (рис. 10) и автомата обработки входных данных и формирования выходных данных A2 Main (рис. 11).

С помощью шаблонов проектирования по диаграммам автомата (рис. 10-11) построены тексты на языке CIAO. Для того чтобы сгенерировать программный код на языке C++, были применены шаблоны реализации [12].

Перед компиляцией в инструментальной среде заготовки программы были доработаны вручную: переопределены примитивы взаимодействия и синхронизации (многопоточность, механизмы обмена данными, таймеры), соответствующие используемой операционной системе.

В результате трансляции был получен исполняемый модуль автомата-преобразователя.

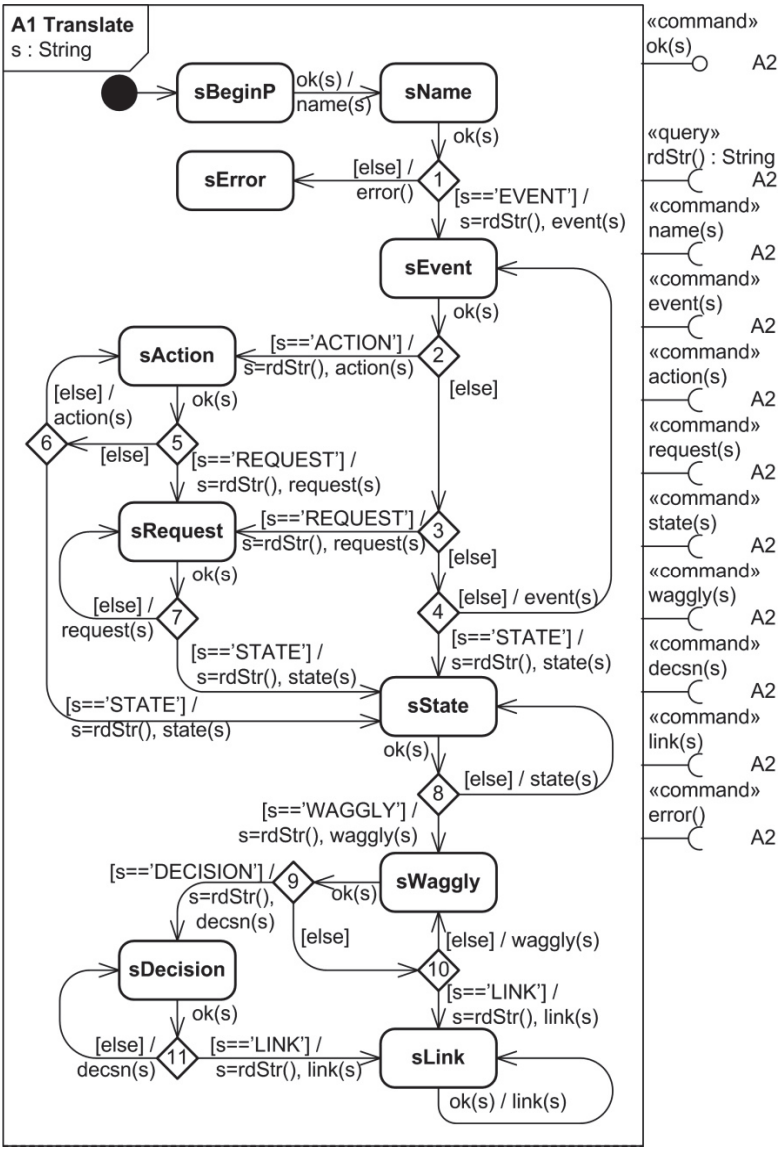

Рис. 10. Автомат-преобразователь A1 Translate

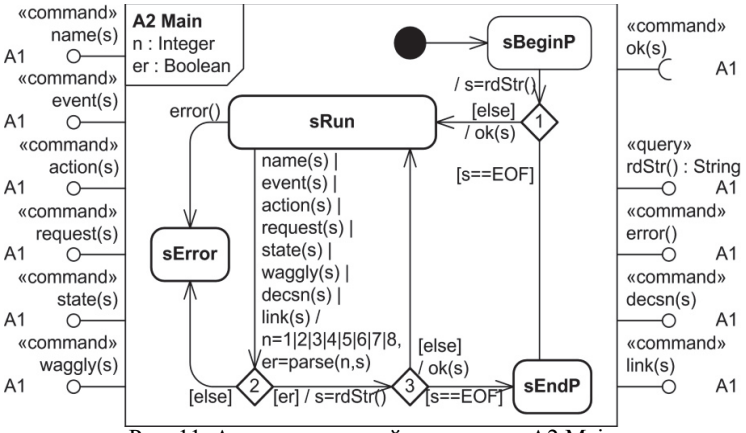

Рис. 11. Автомат головной программы A2 Main

8. Синтез программы управления лифтом. Применим автомат-преобразователь, полученный в разделе 7, к текстовой спецификации системы управления лифтом, полученной в разделе 6. В итоге получаем заготовку реализации автомата A2 Control на языке C++ (листинг 3).

```
typedef enum {
 idle, service
} State2;
class I2 {
public:
 void call(int y);
 void goTo(int y);
};
class A2 : public I2 {
 State2 z;
 I1* outI1;
 I3* outI3;
public:
 void call(int y) {}
 void goTo(int y) {}
 void run() {
  z=idle;
  while (1) {
   event = wait(1);
   switch (z) {
   case idle:
    if (event==call(inF)) {
     if (outI3->isFloor(inF)) {
      outI3->lightOn();
      outI3->open();
      outI1->opened();
      z=service;
     }
     else {
      outI3->move(inF);
      outI3->lightOn();
      outI1->open();
      outI3->opened();
      z=service;
     }
    }
    break;
   case service:
    if (event==timer(15 s)) {
     if (outI3->isEmpty()) {
      outI3->close();
      outI3->lightOff();
      z=idle;
     }
     else {
      outI1->opened();
      z=service;
     }
    }
    if (event==goTo(outF)) {
     if(outI3->isFloor(outF)){
      outI3->opened();
      z=service;
     }
     else {
      if (outI3->isOk()) {
       outI3->close();
       outI3->move();
       outI3->open();
       outI1->opened();
       z=service;
      } ...
```

Листинг 3. Заготовка кода на языке C++ системы управления лифтом

Приведенный в листинге 3 программный код получен полностью автоматически. После этого перед компиляцией был вручную переработан механизм вызова событий (строки 21, 25, 44, 55 листинга 3): использовался класс команд, который содержит номер этажа и тип команды (CallCmd или GotoCmd), и очередь таких команд.

9. Сопоставление предлагаемой методики с другими методами. Рассмотрим наиболее значимые подходы к описанию поведения реагирующих систем.

Метод Statemate и диаграммы состояний Харела. Идеи использования графов переходов состояний как средства описания алгоритмов и моделирования поведения были развиты и адаптированы Д. Харелом [4] в качестве визуальной нотации для больших и сложных реагирующих систем. Диаграммы Харела используются в качестве языка спецификации поведения реагирующих систем в инструментальных средствах I-Logix Statemate [22], IBM Rational Rhapsody [23], AnyLogic [24] и MathWorks Stateflow [25].

Метод Statemate позволяет быстро создать программную систему, для этого необходимо построить три представления:

1) функциональное, в виде диаграмм деятельности;

2) поведенческое, в виде диаграмм состояний;

3) структурное, в виде диаграммы модулей.

Д. Харел впервые показал, что строгое описание поведения является главным элементом в разработке реагирующих систем и служит основой для всего: от спецификации требований до документации пользователя.

Однако некоторые расширения являются избыточными и усложняют использование нотации. К недостаткам метода также можно отнести громоздкость диаграмм из-за использования словесных обозначений и отсутствие некоммерческих инструментальных средств поддержки.

Методология разработки систем реального времени ROOM. Объектно-ориентированная методология ROOM (Real-Time Object-Oriented Modeling) [5] предназначена для проектирования систем реального времени. Поведение программной системы отражено на диаграмме ROOMchart, которая является расширением диаграммы Харела. Основными элементами методологии ROOM являются активные объекты (actors), которые связаны между собой «портами» и инкапсулируют параллельное поведение. Обмен сообщениями происходит как в синхронном, так и в асинхронном режиме. Порядок обмена описывают в «протоколах» и диаграммах последовательности.

Модель системы определена тремя независимыми представлениями:

1) структурной диаграммой компонентов активных объектов и их связей в терминах портов и «соединителей» (connectors);

2) поведенческой диаграммой состояний;

3) иерархической диаграммой классов.

Методология ROOM лежит в основе коммерческого средства ObjecTime Developer [26]. Созданные в нем модели можно рассматривать как ранний исполняемый прототип разрабатываемой системы. В настоящее время методология поддерживается проектом Eclipse eTrice [27]. Инструмент eTrice предоставляет текстовый и графический редакторы, генераторы кода для языков Java, C++ и C, и промежуточное программное обеспечение. Идеи метода также были адаптированы для использования с UML и воплощены в языке UML-RT [28-31], поддержка которого реализована в коммерческих инструментах Rose RT [32], RSA-RTE [33] и проекте Papyrus-RT [34-37].

Идея выделения активных объектов, поддержка текстового варианта языка, быстрое прототипирование и наличие доступного средства eTrice выгодно отличает этот подход от других. Благодаря использованию объектной ориентации и исполняемых моделей генерация кода происходит быстро и эффективно. Применение портов, соединителей и протоколов обеспечивает подробное определение сложных двунаправленных интерфейсов, что можно отнести к преимуществам данной методологии. Соединители обеспечивают статическое и динамическое связывание портов.

Однако присутствие портов и их обозначений на диаграмме сильно ухудшает наглядность. Отметим также, что применение ROOM оправдано только при проектировании больших систем реального времени, для небольших систем такие модели громоздки и избыточны. К тому же программный код, полученный с помощью eTrice, нуждается в промежуточном программном обеспечении.

Метод архитектурного проектирования и моделирования параллельных объектов COMET и язык UML. Объектно-ориентированный метод COMET (Concurrent (Collaborative) Object Modeling and Architectural Design Method) [6, 7] предназначен для проектирования параллельных программных систем, в том числе распределенных приложений и систем реального времени. Для моделирования используется язык UML.

Системы проектируются как гетерогенные подсистемы, работающие параллельно. Использование понятия

«подсистема» (subsystem) выгодно отличает метод COMET от других объектно-ориентированных методологий разработки. Подсистема реализуется в виде набора параллельных задач на одной вычислительной платформе. Взаимодействие подсистем происходит через обмен сообщениями и может осуществляться как синхронно, так и асинхронно.

Основные этапы моделирования:

1. Разработка модели требований: описание функциональных требований к системе в терминах действующих лиц и вариантов использования, описание каждого варианта использования, создание временных прототипов системы для уточнения требований.

2. Аналитическое моделирование: построение статической (диаграммы классов, выделение внешних классов) и динамической моделей системы (диаграммы последовательности или кооперации, диаграммы состояний), детализация вариантов использования.

3. Разработка проектной модели: построение архитектуры путем отображения аналитической модели на среду эксплуатации, определение параллельных задач, проектирование интерфейсов, разбиение системы на подсистемы (диаграмма пакетов).

Несмотря на имеющееся подробное изложение применения метода с примерами [6], на практике выполнить все этапы жизненного цикла даже для небольших проектов оказывается непосильной задачей. По сравнению с ROOM метод COMET является более гибким, но в то же время более сложным в использовании.

Автоматное программирование (SWITCH-технология). Для разработки широкого класса систем управления, реагирующих на события, в том числе программного обеспечения сложных технических устройств, в [8] предложена парадигма автоматного программирования, также известная под названием «SWITCH-технология». В ходе развития SWITCH-технологии были разработаны средства интеграции автоматных моделей с объектно-ориентированными технологиями [38]. Подход состоит в представлении и реализации программ со сложным поведением как *систем автоматизированных объектов управления*. Каждый автоматизированный объект является совокупностью системы управления и объекта управления. Система управления может состоять из одного или нескольких модернизированных конечных автоматов, взаимодействующих между собой. Основной целью разделения системы на объекты и системы управления является попытка отделить логику поведения (управляющие состояния) от количественных результатов действий (вычислительные состояния).

Процесс проектирования программы в виде системы автоматизированных объектов в рамках объектно-ориентированного подхода можно описать следующим образом:

1) построение словесного описания проектируемой системы, выделение взаимодействующих сущностей (объектная декомпозиция), спецификация событий и объектов управления;

2) построение схемы связей для каждого выбранного объекта со сложным поведением, формализация списка входных и выходных воздействий на схеме связей;

3) построение управляющего автомата для каждого объекта на основе схемы связей: выделение управляющих состояний, определение переходов между ними, определение входных событий, входных и выходных переменных;

4) реализация классов и интерфейсов на выбранном языке программирования либо автоматически, либо вручную с помощью шаблонов проектирования.

Метод и средство для моделирования и реализации объектно-ориентированных программ с явным выделением состояний реализованы в рамках открытого проекта UniMod [38, 39].

Основные достоинства автоматного подхода:

– формально и понятно описывает поведение;

– эффективен для систем со сложным поведением;

– позволяет быстро разработать программу в целом;

– схема связей наглядно показывает, какие воздействия являются входными, а какие – выходными;

– обеспечивает переход от неформальных определений автоматов, входных и выходных воздействий к формальным идентификаторам, что дает возможность автоматической генерации кода;

– определяет состав и содержание проектной документации.

– Для реагирующих систем технология автоматного программирования обладает наибольшей эффективностью, однако этот подход имеет следующие недостатки:

– недостаточно проработаны вопросы асинхронного и параллельного взаимодействия объектов;

– при большой размерности задачи необходимость поддержки перечня формальных идентификаторов может оказаться довольно трудоемким процессом, краткие символьные названия могут усложнить дальнейшую модификацию программ;

– на переходе нельзя чередовать условия и действия;

– переходы по условию могут содержать проверку установки определенных значений, что требует постоянного опроса переменных;

– в настоящее время проект UniMod не поддерживается.

Прямое сравнение базовых концепций предлагаемой методики с другими методами показывает, что данная методика никоим образом не противоречит предшествующим разработкам. Напротив, мы используем все наилучшие известные практики. Основное отличие предлагаемой методики от других состоит в целях, которые были поставлены. Главная цель – сократить удельные индивидуальные трудозатраты на разработку при сохранении приемлемого качества продукта. Поэтому, по возможности, упростили лучшие известные практики для ограниченного класса систем – гетерогенных реагирующих систем среднего масштаба и добавили применение средств, хорошо зарекомендовавших себя в этой области: прежде всего, использование регулярных контекстно-свободных грамматик для анализа входного и синтеза выходного кода, а также использование алгоритмов анализа путей в графах переходов состояний для проверки свойств конструируемых систем, в частности для проверки выполнения требований. Поставленная цель достигнута – трудозатраты на разработку и отладку программных систем получения и передачи астрономических данных от различных типов (в том числе высокоскоростных) фотоприемных устройств сократились при одновременном повышении надежности [2, 12].

10. Заключение. Описана методика применения языка CIAO для проектирования и реализации реагирующих гетерогенных систем на основе сетей взаимодействующих автоматных объектов. Особенностью данной разработки является совместное использование механизмов трех концептуальных уровней: графического моделирования сложного поведения с помощью модифицированных диаграмм языка UML, синтаксически-управляемого перевода на основе регулярных контекстно-свободных грамматик, шаблонов автоматической генерации кода на языке C++ с набором инструментальных средств для создания и отладки всех компонентов конструируемой программы.

Построенная автоматная модель таких расширенных диаграмм реализована в языке CIAO и обеспечивает модульность описания поведения программной системы, а также задает интерфейс взаимодействия как автоматов, так и любых других программных компонентов. Язык CIAO превосходит известные аналоги в части предоставления механизмов кооперативного взаимодействия, асинхронности и параллельности для рассматриваемого класса реагирующих систем.

Дальнейшая работа будет заключаться в расширении функциональности созданных компонентов по трем направлениям: 1) уточнение графической нотации графов переходов состояний с

целью получения большей выразительности и наглядности исходных спецификаций; 2) построение средств автоматизированной проверки свойств построенных графов переходов состояний; 3) расширение возможностей кодогенератора и программы имитационного моделирования сгенерированного кода.

Литература

1. *Harel D.* Statecharts: a Visual Formalism for Complex Systems // Science of Computer Programming. 1987. vol. 8. pp. 231–274.
2. *Афанасьева И.В., Новиков Ф.А.* Архитектура программного обеспечения систем оптической регистрации // Информационно-управляющие системы. 2016. № 3. С. 51–63.
3. *Levonevskiy D., Novikov F., Fedorchenko L., Afanasieva I.* Verification of Internet Protocol Properties Using Cooperating Automaton Objects // Proceedings of the 12th International Conference on Security of Information and Networks (SIN'19). ACM. 2019. pp. 1–4.
4. *Harel D., Naamad A.* The STATEMATE semantics of statecharts // ACM Transactions on Software Engineering and Methodology (TOSEM). 1996. vol. 5. no. 4. pp. 293–333.
5. *Selic B., Gullekson G., Ward P.T.* Real-Time Object-Oriented Modeling // John Wiley & Sons. 1994. 525 p.
6. *Gomaa H.* Designing Concurrent, Distributed, and Real-Time Applications with UML // Proceedings of the 28th international conference on Software engineering. 2006. pp. 1059–1060.
7. *Gomaa H.* Real-Time Software Design for Embedded Systems // Cambridge University Press. 2016. 586 p.
8. *Поликарпова Н.И., Шалыто А.А*. Автоматное программирование // Питер. 2011. 176 с.
9. *Новиков Ф.А., Тихонова У.Н.* Автоматный метод определения проблемно-ориентированных языков (Часть 3) // Информационно-управляющие системы. 2010. № 3. С. 29–37.
10. *Fedorchenko L., Baranov S.* Equivalent Transformations and Regularization in Context–Free Grammars // Cybernetics and Information Technologies. 2015. vol. 14. no. 4. pp. 29–44.
11. *Новиков Ф.А., Иванов Д.Ю*. Моделирование на UML. Теория, практика, видеокурс // Наука и Техника. 2010. 640 с.
12. *Афанасьева И.В.* Метод проектирования и реализации параллельных реагирующих систем: диссертация // Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики. 2018. 137 с.
13. *Knuth D.E.* The Art of Computer Programming: Fundamental Algorithms. 3rd ed. // Addison-Wesley Professional. 1998. vol. 1. 652 p.
14. *Novikov F. et al.* Attribute-based approach of defining the secure behavior of automata objects // Proceedings of the 10th International Conference on Security of Information and Networks (SIN'17). 2017. pp. 67–72.
15. *Новиков Ф.А., Афанасьева И.В*. Кооперативное взаимодействие автоматных объектов // Информационно-управляющие системы. 2016. № 6. С. 50–63.
16. *Meyer B.* Object-Oriented Software Construction: 2nd ed. // Prentice-Hall. 1997. 1296 p.
17. *Nesteruk D.* Design Patterns in .NET: Reusable Approaches in C# and F# for Object-Oriented Software Design // Apress. 2019. 376 p.

18. OMG. Unified Modeling Language. Ver. 2.5.1. URL: http://www.omg.org/spec/UML (дата обращения: 01.03.2020).
19. *Yourdon E.* Modern structured analysis // Prentice Hall. 1989. 688 p.
20. *Мартыненко Б.К.* Синтаксически управляемая обработка данных // Издательство Санкт-Петербургского университета. 2004. 316 с.
21. *Федорченко Л.Н., Афанасьева И.В.* Метод описания систем со сложным поведением на принципах обобщённых автоматов // Вестник Бурятского государственного университета. Математика, информатика. 2018. № 4. С. 22–36.
22. *Harel D.* Modelling Reactive Systems with Statecharts: The Statemate Approach // McGraw-Hill. 1998. 258 p.
23. *Harel D.* The Rhapsody semantics of statecharts // Integration of Software Specification Techniques for Applications in Engineering. 2004. pp. 325–354.
24. *Grigoryev I.* AnyLogic 7 in Three Days: A Quick Course in Simulation Modeling. 2nd ed. // CreateSpace Independent Publishing Platform. 2015. 256 p.
25. MathWorks. Stateflow. URL: https://www.mathworks.com/help/stateflow/index.html (дата обращения: 01.03.2020).
26. *Selic B.* ObjecTime Limited, Real-Time ObjectOriented Modeling (ROOM) // Proceedings of the 2nd IEEE Real-Time Technology and Applications Symposium (RTAS ’96). 1996. pp. 214.
27. *Kanav S., Lúcio L., Hilden C., Schuetz T.* Design and Runtime Verification Side-by-Side in eTrice // NASA Formal Methods Symposium. 2019. pp. 255–262.
28. *Selic B.* Using UML for modeling complex real-time systems // Languages, Compilers, and Tools for Embedded Systems. 1998. pp. 250–260.
29. *Posse E., Dingel J.* An executable formal semantics for UML-RT // Software and Systems Modeling. 2016. vol. 15. pp. 179–217.
30. *Das T.K., Dingel J.* Model development guidelines for UML-RT: conventions, patterns and antipatterns // Software and Systems Modeling. 2018. vol. 17. pp. 717–752.
31. *Kedwan F., Sharma C.* Model-Driven Software Development Platforms Reviews // International Journal of Computer Applications. 2019. vol. 178. no. 31. pp. 24–33.
32. IBM Rational Rose RealTime. URL: https://www.ibm.com/support/pages/node/574287 (дата обращения: 01.03.2020).
33. IBM Rational Software Architect RealTime Edition, v9.5.0 Product Documentation. URL: https://www.ibm.com/support/knowledgecenter/SS5JSH_9.5.0 (дата обращения: 01.03.2020).
34. *Hili N., Dingel J., Beaulieu A.* Modelling and Code Generation for Real-Time Embedded Systems with UML-RT and Papyrus-RT // Proceedings of the 39th International Conference on Software Engineering Companion (ICSE-C). IEEE/ACM. 2017. pp. 509–510.
35. *Kahani N., Hili N., Cordy J.R., Dingel J.* Evaluation of UML-RT and Papyrus-RT for modelling self-adaptive systems // Proceedings of the 9th International Workshop on Modelling in Software Engineering. 2017. pp. 12–18.
36. *Kahani N., Bagherzadeh M., Dingel J., Cordy J.R.* The problems with Eclipse modeling tools: A topic analysis of Eclipse forums // Proceedings of the 19th International Conference on Model Driven Engineering Languages and Systems. ACM/IEEE. 2016. pp. 227–237.
37. Eclipse Papyrus Real Time. URL: https://www.eclipse.org/papyrus-rt (дата обращения: 01.03.2020).
38. *Gurov V.S., Mazin M.A., Narvsky A.S., Shalyto A.A.* Tools for support of automata-based programming // Programming and Computer Software. 2007. vol. 33. no. 6. pp. 343–355.
39. *Ricca F. et al.* On the impact of state-based model-driven development on maintainability: a family of experiments using UniMod // Empir Software Eng. 2018. vol. 23. pp. 1743–1790.

Афанасьева Ирина Викторовна — канд. техн. наук, заведующий лабораторией, лаборатория перспективных разработок, Федеральное государственное бюджетное учреждение науки Специальная астрофизическая обсерватория Российской академии наук (САО РАН). Область научных интересов: технологии реализации предметно-ориентированных языков программирования, параллельное программирование, разработка систем сбора астрономических данных. Число научных публикаций — 16. riv@sao.ru; , 369167, Нижний Архыз, Зеленчукский район, Карачаево-Черкесская республика, Россия; р.т.: +7(878)229-3432.

Новиков Фёдор Александрович — д-р техн. наук, старший научный сотрудник, профессор, высшая школа прикладной математики и вычислительной физики, Федеральное государственное автономное образовательное учреждение высшего образования «Санкт-Петербургский политехнический университет Петра Великого» (СПбПУ). Область научных интересов: формализация моделирования, методы алгоритмизации предметных областей, символический искусственный интеллект. Число научных публикаций — 95. fedornovikov51@gmail.com; ул. Политехническая, 29, 195251, Санкт-Петербург, Россия; р.т.: +7(921)574-7944.

Федорченко Людмила Николаевна — канд. техн. наук, старший научный сотрудник, лаборатория прикладной информатики и проблем информатизации общества, Федеральное государственное бюджетное учреждение науки Санкт-Петербургский институт информатики и автоматизации Российской академии наук (СПИИРАН); доцент, кафедра информатики математико-механического факультета, Санкт-Петербургский государственный университете (СПбГУ). Область научных интересов: синтаксически управляемая обработка данных, регуляризация трансляционных грамматик, применение синтаксических методов в прикладных задачах, технология реализации языков программирования. Число научных публикаций — 70. lnf@iias.spb.su; 14 линия В.О., 39, 199178, Санкт-Петербург, Россия; р.т.: +79211838657.

DOI 10.15622/sp.2020.19.3.1

I. AFANASIEVA, F. NOVIKOV, L. FEDORCHENKO

METHODOLOGY FOR DEVELOPMENT OF EVENT-DRIVEN SOFTWARE SYSTEMS USING CIAO SPECIFICATION LANGUAGE

Afanasieva I., Novikov F., Fedorchenko L. **Methodology for Development of Event-driven Software Systems using CIAO Specification Language.**

Abstract. Event-driven software systems, belonging to the class of systems with complex behavior in the scientific literature, are reactive systems, which react to the same input effect in different ways depending on their state and background.

It is convenient to describe such systems using state-transition models utilizing special language tools, both graphical and textual. Methodology for automated development of systems with complex behavior using the designed CIAO language (Cooperative Interaction of Automata Objects), which allows formally specifying the required behavior based on an informal description of the reacting system, is presented.

An informal description of a reacting system can be provided verbally in a natural language or in another way adopted in a specific domain. Further, according to this specification in the CIAO language, a software system for interacting automata in the C++ programming language is generated with a special system.

The generated program implements a behavior guaranteed to correspond to a given specification and original informal description. CIAO provides both graphical and textual notation. Graphic notation is based on an extended notation of state machine diagrams and component diagrams of the unified modeling language UML, which are well established in describing the behavior of event-driven systems.

The text syntax of the CIAO language is described by context-free grammar in regular form. Automatically generated C++ code allows using of both library and any external functions written manually.

At the same time, the evident correspondence of the formal specification and the generated code is preserved on conditions that the external functions conform to their specifications.

As an example, an original solution to D. Knut's problem of a responsive elevator control system is proposed. The effectiveness of the proposed methodology is demonstrated, since the automaton-converter generating the C++ code is presented as a responsive system, is specified in the CIAO language and implemented by the bootstrapping. The proposed methodology is compared with other well-known formal methods for describing systems with complex behavior.

Keywords: Behavior Model, Systems with Complex Behavior, Reactive Systems, State Transition Graph, Syntactic Flow-chart (Graph-scheme), Context-free Grammar in Regular Form, C++ Code Generation Patterns.

Afanasieva Irina — Ph.D., Head of Laboratory, Advanced Design Laboratory, Special Astrophysical Observatory of the Russian Academy of Sciences (SAO RAS). Research interests: implementation technologies for domain-specific programming languages, concurrent programming, development of astronomical data acquisition systems. The number of publications — 16. riv@sao.ru; 369167, Nizhnij Arkhyz, Zelenchukskiy area, Karachay-Cherkessian Republic, Russia; office phone: +7(878)229-3432.

Novikov Fedor — Ph.D., Professor, Professor, Higher School of Applied Mathematics and Computational Physics, Peter the Great St. Petersburg Polytechnic University (SPbPU). Research interests: formalization of modeling, subject area algorithms, symbolic artificial intelligence. The number of publications — 95. fedornovikov51@gmail.com; 29, Politehnicheskaya str., 195251, Russia; office phone: +7(921)574-7944.

Fedorchenko Ludmila — Ph.D., Senior Researcher, Laboratory of Applied Informatics and Society Informatization Problems, St. Petersburg Institute for Informatics and Automation of Russian Academy of Sciences (SPIIRAS); Associate Professor, Department of Informatics of Faculty of Mathematics and Mechanics, Saint Petersburg State University (SPbU). Research interests: syntactically controlled data processing, regularization of translational grammars, application of syntactic methods in applied problems, technology for implementing programming languages. The number of publications — 70. lnf@iias.spb.su; 39, 14-th Line V.O., 199178, Russia; office phone: +79211838657.

References

1. Harel D. Statecharts: a Visual Formalism for Complex Systems. *Science of Computer Programming*. 1987. vol. 8. pp. 231–274.
2. Afanasieva I.V., Novikov F.A. [Software Architecture of Optical Detector Systems]. *Informatsionno-upravliaiushchie sistemy – Information and Control Systems.* 2016. vol. 3. pp. 51–63. (In Russ.).
3. Levonevskiy D., Novikov F., Fedorchenko L., Afanasieva I. Verification of Internet Protocol Properties Using Cooperating Automaton Objects. Proceedings of the 12th International Conference on Security of Information and Networks (SIN'19). ACM. 2019. pp. 1–4.
4. Harel D., Naamad A. The STATEMATE semantics of statecharts. *ACM Transactions on Software Engineering and Methodology (TOSEM)*. 1996. vol. 5. no. 4. pp. 293–333.
5. Selic B., Gullekson G., Ward P.T. Real-Time Object-Oriented Modeling. John Wiley & Sons. 1994. 525 p.
6. Gomaa H. Designing Concurrent, Distributed, and Real-Time Applications with UML. Proceedings of the 28th international conference on Software engineering. 2006. pp. 1059–1060.
7. Gomaa H. Real-Time Software Design for Embedded Systems. Cambridge University Press. 2016. 586 p.
8. Polikarpova N.I., Shalyto A.A. *Avtomatnoe programmirovanie* [Automata-Based Programming]. Piter. 2011. 176 p. (In Russ.).
9. Novikov F.A., Tikhonova U.N. [An Automata Based Method for Domain Specific Languages Definition (Part 3)]. *Informatsionno-upravliaiushchie sistemy – Information and Control Systems.* 2010. vol. 3. pp. 29–37. (In Russ.).
10. Fedorchenko L., Baranov S. Equivalent Transformations and Regularization in Context–Free Grammars. *Cybernetics and Information Technologies.* 2015. vol. 14. no. 4. pp. 29–44.
11. Novikov F.A., Ivanov D.Iu. *Modelirovanie na UML. Teoriia, praktika, videokurs* [Modeling in UML. Theory, Practice, Video Course]. Nauka i Tekhnika. 2010. 640 p. (In Russ.).
12. Afanasieva I.V. *Metod proektirovaniya i realizacii parallel'nyh reagiruyushchih sistem: dissertaciya* [A method for designing and implementing concurrent reactive systems: PhD (tech) thesis]. 2018. 137 p. (In Russ.).
13. Knuth D.E. The Art of Computer Programming: Fundamental Algorithms. 3rd ed. Addison-Wesley Professional. 1998. vol. 1. 652 p.
14. Novikov F. et al. Attribute-based approach of defining the secure behavior of automata objects. Proceedings of the 10th International Conference on Security of Information and Networks (SIN'17). 2017. pp. 67–72.
15. Novikov F.A., Afanasieva I.V. [Cooperative Interaction of Automata Objects]. *Informatsionno-upravliaiushchie sistemy – Information and Control Systems.* 2016. vol. 6. pp. 50–63. (In Russ.).
16. Meyer B. Object-Oriented Software Construction. 2nd ed. Prentice-Hall. 1997. 1296 p.
17. Nesteruk D. Design Patterns in .NET: Reusable Approaches in C# and F# for Object-Oriented Software Design. Apress. 2019. 376 p.

18. OMG. Unified Modeling Language. Ver. 2.5.1. Available at: http://www.omg.org/spec/UML (accessed: 01.03.2020).
19. Yourdon E. Modern structured analysis. Prentice Hall. 1989. 688 p.
20. Martynenko B.K. *Sintaksicheski upravlyaemaya obrabotka dannykh* [Syntactically Controlled Data Processing]. Izdatel'stvo Sankt-Peterburgskogo universiteta. 2004. 316 p. (In Russ.).
21. Fedorchenko L.N., Afanasyeva I.V. [A method for describing systems with complex behavior on the principles of generalized automata]. *Vestnik Buryatskogo gosudarstvennogo universiteta. Matematika, informatika – Bulletin of the Buryat State University. Mathematics, computer science.* 2018. vol. 4. pp. 22–36. (In Russ.).
22. Harel D. Modelling Reactive Systems with Statecharts: The Statemate Approach. McGraw-Hill. 1998. 258 p.
23. Harel D. The Rhapsody semantics of statecharts. Integration of Software Specification Techniques for Applications in Engineering. 2004. pp. 325–354.
24. Grigoryev I. AnyLogic 7 in Three Days: A Quick Course in Simulation Modeling. 2nd ed. CreateSpace Independent Publishing Platform. 2015. 256 p.
25. MathWorks. Stateflow. Available at: https://www.mathworks.com/help/stateflow/index.html (accessed: 01.03.2020).
26. Selic B. ObjecTime Limited, Real-Time ObjectOriented Modeling (ROOM). Proceedings of the 2nd IEEE Real-Time Technology and Applications Symposium (RTAS '96). 1996. pp. 214.
27. Kanav S., Lúcio L., Hilden C., Schuetz T. Design and Runtime Verification Side-by-Side in eTrice. NASA Formal Methods Symposium. pp. 255–262.
28. Selic B. Using UML for modeling complex real-time systems. Languages, Compilers, and Tools for Embedded Systems. 1998. pp. 250–260.
29. Posse E., Dingel J. An executable formal semantics for UML-RT. *Software and Systems Modeling.* 2016. vol. 15. pp. 179–217.
30. Das T.K., Dingel J. Model development guidelines for UML-RT: conventions, patterns and antipatterns. *Software and Systems Modeling.* 2018. vol. 17. pp. 717–752.
31. Kedwan F., Sharma C. Model-Driven Software Development Platforms Reviews. *International Journal of Computer Applications.* 2019. vol. 178. no. 31. pp. 24–33.
32. IBM Rational Rose RealTime. Available at: https://www.ibm.com/support/pages/node/574287 (accessed: 01.03.2020).
33. IBM Rational Software Architect RealTime Edition, v9.5.0 Product Documentation. Available at: https://www.ibm.com/support/knowledgecenter/SS5JSH_9.5.0 (accessed: 01.03.2020).
34. Hili N., Dingel J., Beaulieu A. Modelling and Code Generation for Real-Time Embedded Systems with UML-RT and Papyrus-RT. Proceedings of the 39th International Conference on Software Engineering Companion (ICSE-C). IEEE/ACM. 2017. pp. 509–510.
35. Kahani N., Hili N., Cordy J.R., Dingel J. Evaluation of UML-RT and Papyrus-RT for modelling self-adaptive systems. Proceedings of the 9th International Workshop on Modelling in Software Engineering. 2017. pp. 12–18.
36. Kahani N., Bagherzadeh M., Dingel J., Cordy J.R. The problems with Eclipse modeling tools: A topic analysis of Eclipse forums. Proceedings of the 19th International Conference on Model Driven Engineering Languages and Systems. ACM/IEEE. 2016. pp. 227–237.
37. Eclipse Papyrus Real Time. Available at: https://www.eclipse.org/papyrus-rt (accessed: 01.03.2020).
38. Gurov V.S., Mazin M.A., Narvsky A.S., Shalyto A.A. Tools for support of automata-based programming. *Programming and Computer Software.* 2007. vol. 33. no. 6. pp. 343–355.
39. Ricca F et al. On the impact of state-based model-driven development on maintainability: a family of experiments using UniMod. *Empir Software Eng.* 2018. vol. 23. pp. 1743–1790.