УДК 004.9
DOI: 10.18101/2304-5728-2020-2-15-35

О ПОСТРОЕНИИ СИСТЕМ СО СЛОЖНЫМ ПОВЕДЕНИЕМ НА ПРИНЦИПАХ СИНТАКСИЧЕСКИ ОРИЕНТИРОВАННОГО УПРАВЛЕНИЯ

© Федорченко Людмила Николаевна
кандидат технических наук, старший научный сотрудник,
Санкт-Петербургский институт информатики и автоматизации Российской академии наук (СПИИРАН)
Россия, 199178, г. Санкт-Петербург, 14 Линия В.О., 39
lnf@iias.spb.su

© Афанасьева Ирина Викторовна
кандидат технических наук,
заведующий лабораторией перспективных разработок,
Специальная астрофизическая обсерватория Российской академии наук (САО РАН)
Россия, 369167, пос. Нижний Архыз, Карачаево-Черкесская респ.
riv@sao.ru

Аннотация. К системам со сложным поведением относят событийно-управляемые программные системы, называемые в научной литературе *реагирующими системами* (reactive systems), то есть такими системами, которые на одно и то же входное воздействие реагируют различным образом в зависимости от своего состояния и предыстории. Такие системы удобно описывать с помощью специальных языковых средств, как графических, так и текстовых. В статье представлен подход автоматизированного построения систем со сложным поведением с использованием разработанного языка CIAO (Cooperative Interaction of Automata Objects) [1–2], который позволяет на основе неформального описания реагирующей системы формально специфицировать требуемое поведение. Далее по этой спецификации на языке CIAO генерируется программная система на языке программирования C++. Для языка CIAO предусмотрена как графическая, так и текстовая нотация. Графическая нотация основана на расширенной нотации диаграмм компонентов языка 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. В качестве примера предложено оригинальное решение задачи Д. Кнута о реагирующей системе управления лифтом.

Ключевые слова: граф переходов состояний; синтаксическая граф-схема; грамматика в регулярной форме; конечный автомат.

Для цитирования:
Федорченко Л. Н., Афанасьева И. В. О построении систем со сложным поведением на принципах синтаксически ориентированного управления // Вестник Бурятского государственного университета. Математика, информатика. 2020. № 2. С. 15–35.

Введение

Основная трудность при проектировании и реализации программных и программно-аппаратных прикладных систем со сложным поведением состоит в 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 формальных методов, обеспечивающих качество процесса построения систем.

В данной статье кратко излагается методика разработки асинхронных реагирующих систем [3], 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.

2. По графической спецификации автоматных объектов составляется функционально эквивалентный текст программы на языке CIAO. Этот процесс в настоящее время автоматизирован частично.

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.

1 Историческая справка

Методы разработки сложных программных систем определяются используемой моделью поведения. В известной классификации поведенческих моделей программ Д. Харела выделяются *реагирующие* и *трансформационные системы* [3]. Большинство систем управления техническими средствами являются *реагирующими* системами, в которых 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 *асинхронную схему: Событие → Опросить входы → Вычислить выходы → Изменить состояние → Новое событие.* 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 выросла значительно, 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 [3; 4], методологию разработки систем реального времени ROOM [5], метод архитектурного проектирования и моделирования параллельных объектов COMET [6; 7] и автоматное программирование (SWITCH-технологию) [8].

Методика имеет достаточно богатую предысторию развития и обширные основания. Опираясь на имеющийся практический опыт разработки систем управления [2] и применяя опробованные ранее методы, такие как конструирование специализированных языков предметной области [9], применение синтаксически управляемой обработки данных [10; 19], использование графических средств моделирования предметных областей [11], мы предлагаем методику, продолжающую и развивающую как наши предшествующие разработки, так и разработки коллег.

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].

Цель исследования, которому посвящена данная работа, — 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 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 преимущества.

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*. Каждый отдельный автомат встроен в свой программный объект, который называется *автоматным объектом*. Поведение системы в целом складывается из поведения автоматных объектов, взаимодействующих через определенные интерфейсы.

Автоматные объекты могут выполняться на различных устройствах, поэтому рассматривается случай гетерогенных систем. В этом заключается основная идея и мотивация языка CIAO — описать возможные сценарии поведения сложных гетерогенных и распределенных реагирующих систем.

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*, 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

Для спецификации поведения реагирующих систем предлагается графический язык спецификации CIAO, построенный на основе унифицированного языка моделирования UML[1] [11], с использованием и модификацией некоторых конструкций диаграммы автомата, диаграммы компонентов и диаграммы классов.

Графическая форма язык CIAO предназначен для: спецификации и визуализации сложного поведения; публикации (параллельных) алгоритмов; построения имитационных моделей сложного поведения и исследования этих моделей; быстрого прототипирования реагирующих систем.

Следуя традиции, введенной в практику авторами UML[2] [11], семантика графического языка описывается диаграммой классов, представляющей взаимосвязи между основными понятиями языка (такая диаграмма обычно называется метамоделью), а графическая нотация иллюстрируется примерами.

[1] OMG. Unified Modeling Language. Ver. 2.5.1. URL: http://www.omg.org/spec/UML (дата обращения: 01.03.2020).
[2] Там же.

Для сравнения и указания основных нововведений языка CIAO на рисунке 1 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.

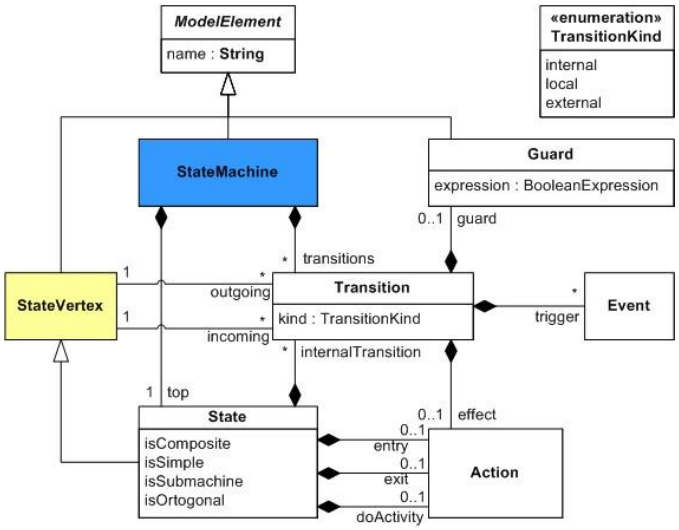

Рис. 1. Метамодель диаграммы автомата UML[3]

Событие должно быть предоставляемой командой, эффект — возможно, несколькими командами либо выражением. Сторожевое условие имеет вид произвольной булевой формулы над требуемыми запросами и значениями локальных переменных. В каждом автоматном объекте присутствуют следующие элементы:

1. Number, name — порядковый номер и имя автоматного объекта.
2. Variables — набор переменных.
3. States — набор состояний.
4. Transitions — набор переходов между состояниями.
5. Interfaces — набор команд и запросов.
6. Links — связи автоматного объекта с компонентами системы.

[3] OMG. Unified Modeling Language. Ver. 2.5.1. URL: http://www.omg.org/spec/UML (дата обращения: 01.03.2020).

4 Формулировка задачи о лифте

В своей фундаментальной монографии Д. Кнут разобрал задачу о проектировании системы управления пассажирским лифтом [13].

Система управления лифтом — пример программы, управляемой событиями. На примере этой задачи покажем возможности 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 если 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.

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. *Все запросы перемещения на этажи и вызовы лифта от этажей в конечном итоге должны быть обслужены.*

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

Таблица 1

Описание переменных и объектов 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)	

Пассажир (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.

Опишем поведение системы управления лифтом (объект Control).

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 с, 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 всё 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 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.

На основании словесных моделей построена система управления лифтом. Работа лифта моделируется с помощью трех компонентов (автоматных объектов), один из которых описывает поведение пассажира (рисунок 2), другой — поведение системы управления лифтом (рисунок 3), а третий описывает работу механизмов лифта (рисунок 4). Эти объекты реализуют все выполняемые действия.

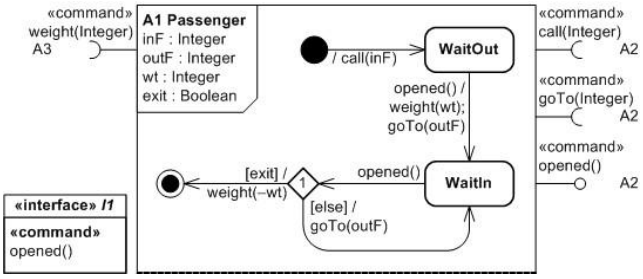

Рис. 2. Автомат A1 (поведение пассажира) и интерфейс I1

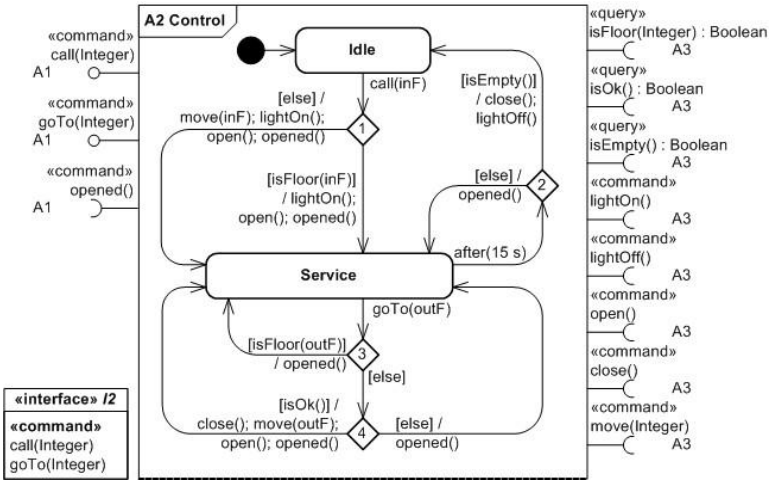

Рис. 3. Автомат A2 (система управления лифтом) и интерфейс I2

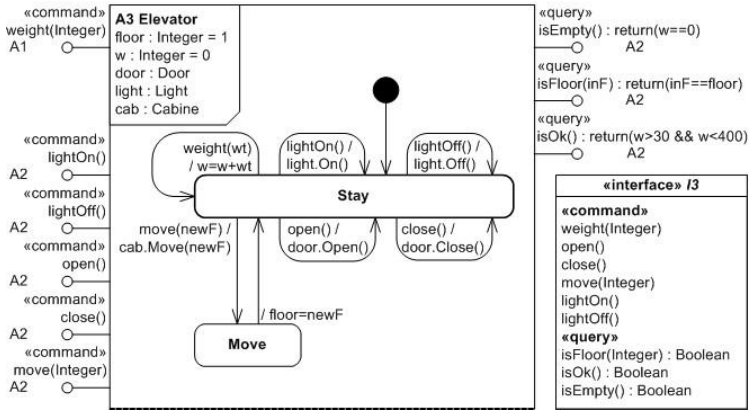

Рис. 4. Автомат A3 (работа механизмов лифта) и интерфейс I3

Для строгой математической проверки того, что полученная спецификация действительно удовлетворяет требованиям на обслуживание, достаточно проследить все возможные пути в графах переходов состояний и убедиться, что они реализуемы и заканчиваются в требуемых состояниях.

5 Методика перевода спецификации CIAO в программу на языке C++

На примере задачи о системе управления лифтом мы 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 нашей целью является не только получение 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) [18], выраженной диаграммой деятельности на языке UML (рисунок 5).

Рис. 5. Схема применения предлагаемой методики

Блоки на этой схеме имеют следующий смысл.

1. **Знания о предметной области**. Для проектирования реагирующей системы необходимы некоторые знания о предметной области системы. В примере с лифтом мы полагаем, что какие-то знания о том, как работает лифт, есть у всех.
2. **Неформальная постановка задачи**. В данном случае задача про лифт неформально описана в начале раздела 4.
3. **Графическая нотация CIAO.** Обозначения на диаграммах, изложенные в разделе 3.
4. **Проектирование**. В результате этого важнейшего творческого, не автоматизируемого *процесса* выделяются автоматные объекты, интерфейсы, состояния и переходы. Всё это оформляется в виде графов переходов состояний, как показано в разделе 4.
5. **Графическая спецификация автоматных объектов**. Это набор диаграмм в нотации языка CIAO, которые в совокупности описывают спецификацию поведения проектируемой системы.
6. **Анализ и верификация.** Важный, но необязательный *процесс,* в результате которого модель итерационно улучшается, и для нее доказываются некоторые свойства. Примеры приведены в разделе 4.
7. **Перевод в текстовый формат**. Формируется грамматическое описание модели поведения реагирующей системы на основе синтаксиса языка CIAO.
8. **Текстовый синтаксис CIAO**. Грамматическое описание текстового синтаксиса языка CIAO приведено в разделе 6.
9. **Специфицирование**. Это полуавтоматический *процесс*, в результате которого строится текстовая спецификация на языке CIAO, как показано в разделе 6.
10. **Текстовая спецификация автоматных объектов**. Это результат перевода спецификации из графической формы в текстовую, пример которой приведен в разделе 6, листинг 2.
11. **Генерация кода**. *Процесс*, в результате которого порождается заготовка исходного кода на языке C++.
12. **Шаблоны генерации кода на C++**. Шаблоны, то есть правила синтаксически управляемого перевода, которые позволяют переводить конструкции языка CIAO в некоторые фрагменты кода на языке C++.
13. **Заготовка кода на языке C++**. Это код на языке C++, реализующий поведение системы. Может содержать вызовы внешних функций, которые должны быть реализованы отдельно от заготовки. Пример приведен в разделе 8, листинг 3.
14. **Внешние функции**. Внешние библиотеки классов и функций, заголовочные файлы либо код функций на языке C++.
15. **Компиляция**. *Процесс* компиляции штатным компилятором C++.
16. **Готовая система**. Готовая к работе система.

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, который непосредственно интерпретирует диаграммы [9].

Однако в практических промышленных приложениях предпочитают получать автоматически исполняемый код на язык программирования на основе спецификации трансляций. Удобно воспользоваться давно проверенными методами синтаксически управляемой обработки данных, использующих контекстно-свободный язык, определяемый 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 некоторой 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 [19].

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 списка семантических процедур.

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

6 Формальная спецификация в текстовой форме

Основные определения даны в работах [10; 20]. 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;

– 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 анализатора (parser) 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   Регулярное_выражение2 |  // Конкатенация (2)
 Регулярное_выражение1 ';' Регулярное_выражение2 |  // Альтернативный выбор (3)
 Регулярное_выражение1 '#' Регулярное_выражение2 |  // Итерация (4)
 '[' 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; 20]. На рисунке 6 показано графическое представление правил грамматики языка CIAO. 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

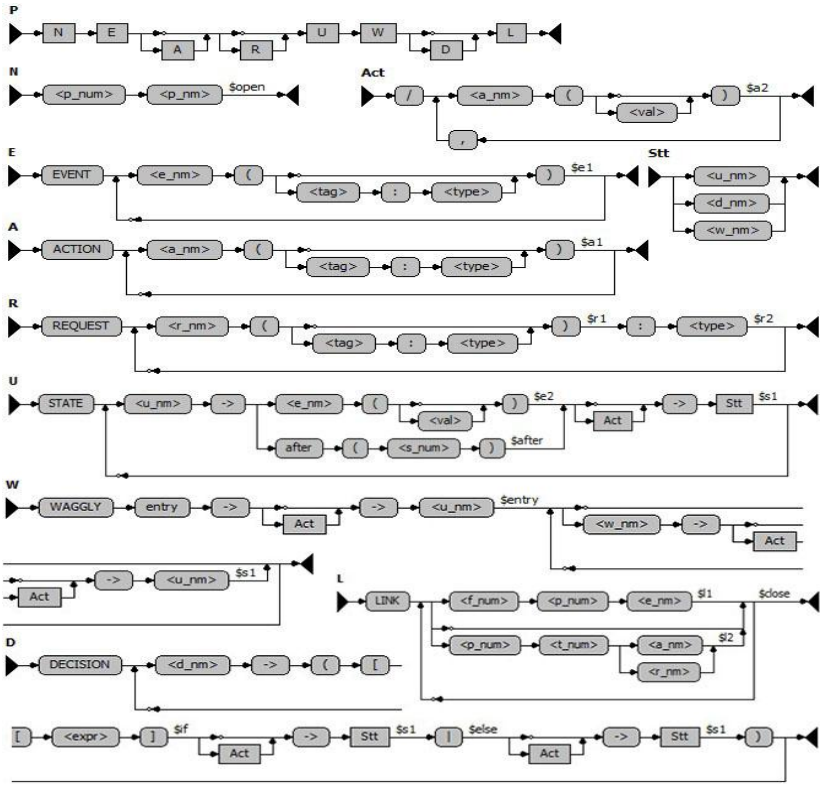

Рис. 6. Синтаксическая граф-схема CIAO

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.

```
#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

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.

7 Автомат-преобразователь — раскрутка языка CIAO

Покажем действенность и работоспособность предлагаемой методики на примере самоприменимости (bootstrapping). А именно, специфицируем на языке CIAO программу, которая переводит спецификации на языке CIAO в программы на языке C++.

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

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 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.

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

3. Поскольку текст программы является промежуточным, считаем, что в нем всё 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).

На основе словесного описания построена система трансляции, состоящая из двух компонентов — автомата-преобразователя A1 Translate и автомата обработки входных данных и формирования выходных данных A2 Main.

С помощью шаблонов проектирования по диаграммам автомата 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.

8 Синтез программы управления лифтом

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

```
1  #2 Control
2  EVENT
3  call(y : Integer)
4  goTo(y : Integer)
5  ACTION
6  opened()
7  lightOn()
8  lightOff()
9  open()
10 close()
11 move(y : Integer)
12 REQUEST
13 isFloor(y : Integer) : Boolean
14 isOk() : Boolean
15 isEmpty() : Boolean
16 STATE
17 idle -> call(inF) -> dcsn1
18 service -> after(15 s) -> dcsn2
19 service -> goTo(outF) -> dcsn3
20 WAGGLY
21 entry -> -> idle
22 DECISION
23 dcsn1 -> ( [isFloor(inF)] /
      lightOn(), open(), opened()
      -> service | / move(inF),
      lightOn(), open(), opened()
      -> service )
24 dcsn2 -> ( [isEmpty()] / close
      (), lightOff() -> idle | /
      opened() -> service )
25 dcsn3 -> ( [isFloor(outF)] /
      opened() -> service | ->
      dcsn4 )
26 dcsn4 -> ( [isOk()] / close(),
      move(outF), open(), opened()
       -> service | / opened() ->
      service )
27 LINK
28 #1 #2 call
29 #1 #2 goTo
30 #2 #1 opened
31 #2 #3 isFloor
32 #2 #3 isOk
33 #2 #3 isEmpty
34 #2 #3 lightOn
35 #2 #3 lightOff
36 #2 #3 open
37 #2 #3 close
38 #2 #3 move
```

Листинг 3. Текстовая спецификация 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.

Заключение

В статье 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.

Прямое сравнение базовых концепций предлагаемой методики с другими методами показывает, что методика, кратко описанная в статье, не противоречит предшествующим разработкам. Напротив, используются все наилучшие известные практики. Основное отличие предлагаемой методики от других состоит в поставленных целях. Главная цель — сократить удельные индивидуальные трудозатраты на разработку при сохранении приемлемого качества продукта.

Язык CIAO превосходит известные аналоги в части предоставления механизмов кооперативного взаимодействия, асинхронности и параллельности для рассматриваемого класса реагирующих систем.

Дальнейшая работа будет состоять в расширении функциональности созданных компонентов по четырем 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; 4) развитие технологии синтаксически ориентированного управления генерацией объектного кода на базе уже существующих инструментальных средств.

Литература

1. Fedor A. Novikov, Ludmila N. Fedorchenko, Vladimir I. Vorobiev, Roza R. Fatkieva, and Dmitriy K. Levonevskiy. 2017. Attribute-Based Approach of Defining the Secure Behavior of Automata Objects. In Proceedings of SIN 2017 conference (SIN 2017), Jennifer B. Sartor, Theo D'Hondt, and Wolfgang De Meuter (Eds.). ACM, New York, NY, USA, Article 4, 6 pages. DOI: https://doi.org/10.1145/3136825.3136887.

2. Новиков Ф.А., Афанасьева И.В. Кооперативное взаимодействие автоматных объектов // Информационно-управляющие системы. 2016. № 6. С. 50–63. DOI: 10.15217/issn1684-8853.2016.6.50.

3. Harel D. Statecharts: a Visual Formalism for Complex Systems // Science of Computer Programming. 1987. Vol. 8. Pp. 231–274. https://doi.org/10.1016/0167-6423(87)90035-9.

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. https://doi.org/10.1145/235321.235322.

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 // Addison-Wesley Professional. 2000. 816 p.

7. Gomaa H. Real-Time Software Design for Embedded Systems // Cambridge. 2016. https://doi.org/10.1017/CBO9781139644532.

8. Поликарпова Н. И., Шалыто А. А. Автоматное программирование. СПб.: 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. https://doi.org/10.1515/cait-2014-0003.

11. Новиков Ф. А., Иванов Д. Ю. Моделирование на UML. Теория, практика, видеокурс. СПб.: Наука и техника. 2010. 640 с.

12. Афанасьева И. В. Метод проектирования и реализации параллельных реагирующих систем: дис. … канд. техн. наук. СПб., 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., Fedorchenko L., Vorobiev V., Fatkieva R., Levonevskiy D. 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). ACM, NY, USA. 2017. Pp. 67–72. https://doi.org/10.1145/3136825.3136887.

15. Новиков Ф. А., Афанасьева И. В. Кооперативное взаимодействие автоматных объектов // Информационно-управляющие системы. 2016. № 6. С. 50–63. https://doi.org/10.15217/issn1684-8853.2016.6.50.

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. https://doi.org/10.1007/978-1-4842-4366-4.

18. Yourdon E. Modern structured analysis // Englewood Cliffs, NJ : Prentice Hall. 1989. 672 p.

19. Мартыненко Б. К. Синтаксически управляемая обработка данных. СПб.: Изд-во С.-Петерб. ун-та, 2004. 316 с.

20. Федорченко Л. Н., Афанасьева И. В. Метод описания систем со сложным поведением на принципах обобщенных автоматов // Вестник Бурятского государственного университета. Математика, информатика. 2018. № 4. С. 22–36. https://doi.org/10.18101/2304-5728-2018-4-22-36.

CONSTRUCTION OF SYSTEMS WITH COMPLEX BEHAVIOR ON THE PRINCIPLES OF SYNTAX-ORIENTED CONTROL

Lyudmila N. Fedorchenko
Cand. Sci. (Engineering), Senior Researcher
St. Petersburg Institute for Informatics and Automation RAS (SPIIRAS),
39 14th Line of V. O., St. Petersburg 199178, Russia
St. Petersburg State University
7/9 Universitetskaya Emb., St. Petersburg 199034, Russia
lnf@iias.spb.su

Irina V. Afanasieva
Cand. Sci. (Engineering), Head of Advanced Developments Laboratory
Special Astrophysical Observatory RAS
Nizhniy Arkhyz 369167, Karachay-Cherkessian Republic, Russia
riv@sao.ru

Abstract. Systems with complex behavior include event-driven software systems, called reactive systems in the academic literature, that is, systems that react to the same input action in different ways depending on their state and history. It is convenient to describe such systems using special language tools, both graphic and text. The article presents a technique for automated construction of systems with complex behavior using the developed language CIAO (Cooperative Interaction of Automata Objects), which allows formally specifying the required behavior based on an informal description of a responsive system. Further according to this specification a software system is generated in the C ++ programming language. Both graphical and textual notations are provided for the CIAO language. Graphic notation is based on an extended notation of state machine diagrams and component diagrams of the unified modeling language UML, which proved their worth in describing the behavior of event-driven systems. The text syntax of the CIAO language is described by a context-free grammar in regular form. The text syntax of the CIAO language is described by context-free grammar in regular form. Automatically generated C++ code allows using both library and any hand-written external functions. As an example, we have proposed an original solution to D. Knuth's problem of a responsive elevator control system.

Keywords: state-transition graph; syntactic flow-chart; finite state transition graph, context-free grammar in regular form.

References

1. Fedor A. Novikov, Ludmila N. Fedorchenko, Vladimir I. Vorobiev, Roza R. Fatkieva, and Dmitriy K. Levonevskiy. 2017. Attribute-Based Approach of Defining the Secure Behavior of Automata Objects. *Proceedings of SIN 2017 Conference (SIN 2017).* Jennifer B. Sartor, Theo D'Hondt, and Wolfgang De Meuter (Eds.). ACM, New York, USA, Art. 4, 6 p. DOI: https://doi.org/10.1145/3136825.3136887.

2. Novikov F. A., Afanasyeva I. V. Kooperativnoe vzaimodeistvie avtomatnykh obyektov [Cooperative interaction of automata objects // Informatsionno-upravlyayushchie sistemy]. *Informatsionno-upravlyayuschie sistemy*. 2016. No. 6. Pp. 50–63. DOI: 10.15217/issn1684-8853.2016.6.50.

3. Harel D. Statecharts: a Visual Formalism for Complex Systems. *Science of Computer Programming.* 1987. V. 8. Pp. 231–274. https://doi.org/10.1016/0167-6423(87)90035-9.

4. Harel D., Naamad A. The STATEMATE semantics of statecharts. *ACM Transactions on Software Engineering and Methodology (TOSEM).* 1996. V. 5. No. 4. Pp. 293–333. https://doi.org/10.1145/235321.235322.

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.* Addison-Wesley Professional, 2000. 816 p.

7. Gomaa H. *Real-Time Software Design for Embedded Systems.* Cambridge, 2016. https://doi.org/10.1017/CBO9781139644532.

8. Polikarpova N. I., Shalyto A. A. *Avtomatnoe programmirovanie* [Automatic Programming]. St. Petersburg: Piter Publ., 2011. 176 p.

9. Novikov F. A., Tikhonova U. N. Avtomatnyy metod opredeleniya problemno-orientirovannykh yazykov (Chast 3) [An Automatic Method for Definition of Object-Oriented Language (Part 3)]. *Informatsionno-upravliaiuschie sistemy — Information and Control Systems*. 2010. No. 3. Pp. 29–37.

10. Fedorchenko L., Baranov S. Equivalent Transformations and Regularization in Context–Free Grammars. *Cybernetics and Information Technologies*. 2015. V. 14. No. 4. Pp. 29–44. https://doi.org/10.1515/cait-2014-0003.

11. Novikov F. A., Ivanov D. Yu. *Modelirovanie na UML. Teoriya, praktika, videokurs* [UML Modeling. Theory, Practice, Video Course]. St. Petersburg: Nauka i Tekhnika, 2010. 640 p.

12. Afanasyeva I. V. *Metod proektirovaniya i realizatsii parallelnykh reagiruyushchikh sistem* [Method of Design and Implementation of Concurrent Reactive Systems]. Cand. Engineering Sci. Diss. St. Petersburg. 2018. 137 p.

13. Knuth D. E. *The Art of Computer Programming: Fundamental Algorithms*. 3rd ed. Addison-Wesley Professional, 1998. V. 1. 652 p.

14. Novikov F., Fedorchenko L., Vorobiev V., Fatkieva R., Levonevskiy D. 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)*. ACM, NY, USA, 2017. Pp. 67–72. https://doi.org/10.1145/3136825.3136887.

15. Novikov F. A., Afanasyeva I. V. Kooperativnoe vzaimodeystvie avtomatnykh obyektov [Cooperative Interaction of Automata Objects]. *Informatsionno-upravliaiuschie sistemy — Information and Control Systems*. 2016. No. 6. Pp. 50–63. https://doi.org/10.15217/issn1684-8853.2016.6.50.

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. https://doi.org/10.1007/978-1-4842-4366-4.

18. Yourdon E. *Modern Structured Analysis*. Englewood Cliffs, NJ: Prentice Hall, 1989. 672 p.

19. Martynenko B. K. *Sintaksicheski upravlyaemaya obrabotka dannykh* [Syntax-Directed Data Processing]. St. Petersburg: St. Petersburg Univ. Publ., 2004. 316 p.

20. Fedorchenko L. N., Afanasyeva I. V. Metod opisaniya sistem so slozhnym povedeniem na printsipakh obobshchennykh avtomatov [A Method for Describing Systems with Complex Behavior Based on the Principles of Generalized Automata]. *Vestnik Buryatskogo gosudarstvennogo universiteta. Matematika, informatika*. 2018. No. 4. Pp. 22–36. https://doi.org/10.18101/2304-5728-2018-4-22-36.