ИПМ им.М.В.Келдыша РАН

Абрау-2017 • Труды конференции

В.И. Шелехов

**Технология
модельно-ориентированного
автоматного программирования**

Рекомендуемая форма библиографической ссылки

Шелехов В.И. Технология модельно-ориентированного автоматного программирования // Научный сервис в сети Интернет: труды XIX Всероссийской научной конференции (18-23 сентября 2017 г., г. Новороссийск). — М.: ИПМ им. М.В.Келдыша, 2017. — С. 468-476. — URL: <http://keldysh.ru/abrau/2017/42.pdf> doi:[10.20948/abrau-2017-42](https://doi.org/10.20948/abrau-2017-42)

Размещена также [презентация к докладу](#)

Технология модельно-ориентированного автоматного программирования

В.И. Шелехов

Институт Систем Информатики им. А.П. Ершова СО РАН, Новосибирск

Аннотация. В технологии автоматного программирования определяется метод построения программ на базе модели с фиксированным управлением и произвольными незаполненными позициями для операторов и данных. Модель автоматной программы строится в виде набора слоев в аспектно-ориентированном стиле. Модель представляется на языке требований в рамках технологии автоматного программирования. Определены модели, обеспечивающие мониторинг и отказоустойчивость контроллера системы управления.

Ключевые слова: модельно-ориентированное проектирование, автоматное программирование, система управление, инженерия требований

Автоматное программирование [1–5] ориентировано на класс *реактивных систем*, реализующих взаимодействие с внешним окружением системы и реагирующих на определенный набор событий (сообщений). Определение требований – первый этап в разработке реактивной системы. Второй этап – формализация функциональных требований, содержательно сформулированных на первом этапе, в виде автоматной программы. Наряду с операторным языком автоматного программирования имеется эквивалентный *язык требований* [2–4], позволяющий проводить компактную формализацию функциональных требований в виде набора правил, по структуре близких к содержательным требованиям.

В настоящей работе рассматривается подход к разработке автоматной программы на базе ее модели, в которой фиксирована лишь управляющая часть программы, а информационная часть схематично представлена набором произвольных данных и операторов, которые предстоит конкретизировать в автоматной программе. Фактически модель является шаблоном для написания программы. Данный метод построения программы на базе ее модели рассматривается в наших работах [3, 5] для контроллеров систем управления.

Модель составлена из набора *слоев* (аспектов). В простейшем случае слой может быть реализован независимой утилитой с предоставлением определенного интерфейса. В общем случае включение в модель очередного слоя требует существенной модификации исходной модели. Отношение включения в модель очередного слоя определяет древесную структуру

множества моделей автоматных программ. Для контроллера системы управления простейшей является модель системы управления в виде бесконечного цикла. Ее можно расширить слоями, обеспечивающими механизмы отказоустойчивости и мониторинга. Цель настоящей работы – описать эти базисные модели для контроллеров систем управления.

Предлагаемый метод разработки автоматных программ на базе моделей (шаблонов управления) является новым направлением *модельно-ориентированной* разработки программ (*model-driven engineering*) [8-10, 12, 13], особенно популярной последние три года. Модельно-ориентированный подход применяется в системной и программной инженерии. Он позволяет существенно сократить сроки разработки сложных программных комплексов. В общепринятой трехуровневой схеме разработки (требования, дизайн, программа) модель строится для дизайна программы, обычно на одном из графических языков: UML, Simulink и многих других. Среди тестовых языков следует отметить семейство языков Epsilon [14]. Для моделирования, верификации, анализа и трансформации модели предоставляется обширный набор инструментов. Код программы генерируется из модели автоматически. В нашем походе применяется двухуровневая схема (требования, автоматная программа). Модель определяется на том же языке автоматного программирования, и поэтому автоматическая генерация кода из модели не нужна. Наша модель является шаблоном автоматной программы, тогда как модель в виде дизайна является эквивалентом кода программы, хотя и определена на языке более высокого уровня.

1. Автоматное программирование

В соответствии с системой классификации программ [6] автоматное программирование ориентировано на класс реактивных систем, реализующих взаимодействие с внешним окружением системы и реагирующих на определенный набор событий (сообщений). Автоматная программа определяет конечный автомат в виде гиперграфа. Вершина автомата – *управляющее состояние* (метка) программы. Ориентированная гипердуга автомата соответствует некоторому *сегменту кода* и связывает одну вершину с одной или несколькими другими вершинами.

Язык автоматного программирования [2] построен как расширение языка предикатного программирования P [7], используемого для построения сегментов кода.

Гиперфункции. Гиперграфовая структура автоматной программы является продолжением аппарата гиперфункций.

Программа $A(x, y)$ с аргументами x и результатами y записывается в виде $A(x: y)$. *Гиперфункция* – программа с несколькими *ветвями* результатов. Гиперфункция $A(x: y: z)$ имеет две ветви результатов y и z . Исполнение гиперфункции завершается одной из ветвей с вычислением результатов по этой ветви; результаты других ветвей не вычисляются.

Вызов гиперфункции записывается в виде $A(x: y \#M1: z \#M2)$. Здесь $M1$ и $M2$ – метки программы, содержащей вызов. Операторы перехода $\#M1$ и $\#M2$ встроены в ветви вызова. Исполнение вызова либо завершается первой ветвью с вычислением y и переходом на метку $M1$, либо второй ветвью с вычислением z и переходом на метку $M2$. Вызов вида $A(x: y \#M1: z \#M2); M1: \dots$ может быть представлен более компактно: $A(x: y: z \#M2)$.

Аппарат *гиперфункций* является более общим и гибким по сравнению с известным механизмом обработки исключений, например, в таких языках, как Java и C++. Использование гиперфункций делает программу короче, быстрее и проще для понимания [2, 11]. Традиционные подходы в реализации обработки аварийных ситуаций предполагают заведение дополнительных структур, усложняющих программу. Этого удастся избежать при использовании гиперфункций. Подробнее в работе [4, разд. 4], подразделе «баланс информационных и управляющих связей».

Автоматная программа определяется следующей конструкцией:

```
process <имя программы>(<описания аргументов и результатов>)  
  { <описания переменных состояния процесса>  
    <сегменты кода>  
  }
```

Состояние автоматной программы определяется значениями набора переменных, модифицируемых в программе, за исключением локальных переменных. Взаимодействие с *внешним окружением* автоматной программы реализуется через прием и посылку *сообщений*, а также через *разделяемые переменные*, доступные в данной программе и других программах из окружения данной программы.

Произвольный <сегмент кода> представляется конструкцией:

<имя управляющего состояния>: <оператор>

Исполнение <оператора> завершается оператором перехода вида $\#M$, реализующим переход на начало сегмента, ассоциированного с управляющим состоянием M . Структура управления автоматной программой аналогична используемой в языке Фортран, применяемом в основном для задач вычислительной математики, но не для реактивных систем.

В качестве примера автоматной программы рассмотрим модуль операционной системы, реализующий следующий сценарий работы с пользователем (рис.1).

Рис. 1 – Схема работы ОС с пользователем: программа и ее автомат

В управляющем состоянии **login** операционная система запрашивает имя пользователя. Если полученное от пользователя имя существует в системе, она переходит в управляющее состояние **passwd**, иначе возвращается в состояние **login**. В состоянии **passwd** система запрашивает пароль. Если поданная пользователем строка соответствует правильному паролю, то пользователь допускается к работе и система переходит в состояние **session**. При завершении работы пользователя система переходит в состояние **login**. Отметим, что реальная программа взаимодействия ОС с пользователем существенно сложнее; в частности, функция `get_string` должна поставлять текст в зашифрованном виде.

В общем случае система управления определяется в виде композиции нескольких независимых автоматных программ, исполняемых параллельно и взаимодействующих между собой через сообщения и разделяемые переменные.

Неблокированный прием сообщения реализуется конструкцией `<имя сообщения>(<параметры>)`. Ее значение – **true**, если из окружения получено сообщение с указанным именем.

Язык требований. Требования — совокупность утверждений относительно свойств разрабатываемой программы. Функциональные требования определяют поведение программы. Их наиболее популярной формой являются *сценарии использования* (*use case*). В нашем подходе они формализованы в виде правил на языке требований. Спецификация на этом языке компактна и легко транслируется в автоматную программу, что позволяет использовать его как язык автоматного программирования.

Требование определяет один из вариантов функционирования автоматной программы. Оно представляется *правилом* следующей структуры:

$$\langle \text{условие}_1 \rangle, \langle \text{условие}_2 \rangle, \dots, \langle \text{условие}_n \rangle \rightarrow \langle \text{действие}_1 \rangle, \dots, \langle \text{действие}_m \rangle;$$

Условиями являются: управляющие состояния, получаемые сообщения, логические выражения. *Действиями* являются: простые операторы, вызовы программ, посылаемые сообщения и итоговые управляющие состояния. Требование является спецификацией некоторого сегмента кода или его части. Семантика требования следующая: если в данный момент времени истинны все условия в левой части требования, то последовательно выполняется набор действий в правой части. Далее ограничимся детерминированными

программами: для исполнения выбирается первое правило с истинными условиями.

Управляющее состояние в качестве <условия> означает утверждение того, что исполнение программы находится в данном управляющем состоянии. Оно должно быть первым в списке условий, и может быть опущено лишь в случае, когда автоматная программа имеет единственное управляющее состояние. Управляющее состояние в качестве <действия> – это следующее управляющее состояние, с которого продолжится исполнение программы после завершения исполнения данного требования. Оно должно быть последним в списке действий.

Для требований следующего вида:

$s1, \langle \text{условие}_1 \rangle, \dots, \langle \text{условие}_n \rangle \rightarrow \langle \text{действие}_1 \rangle, \dots, \langle \text{действие}_m \rangle, s2$

$s1 \rightarrow \langle \text{действие}_1 \rangle, \dots, \langle \text{действие}_m \rangle, s2$

где $s1$ и $s2$ – управляющие состояния, иногда будем использовать, соответственно, другие формы записи требований:

$s1: \langle \text{условие}_1 \rangle, \dots, \langle \text{условие}_n \rangle \rightarrow \langle \text{действие}_1 \rangle, \dots, \langle \text{действие}_m \rangle \#s2$

$s1: \langle \text{действие}_1 \rangle, \dots, \langle \text{действие}_m \rangle \#s2$

В качестве примера рассмотрим требования для модуля, реализующего сценарий работы ОС с пользователем на рис.1.

Содержательное описание представлено выше.

Окружение.

`message Get(string str); // получение строки от пользователя`

Управляющие состояния: login, passw, session;

Требования.

login: `Get(str) → User(str : #login : #passw);`

passw: `Get(str) → Password(str : #passw : #session);`

session: `UserSession() #login`

Здесь `User` и `Password` – гиперфункции с двумя ветвями без результирующих переменных.

2. Базисные модели системы управления

Система управления реализует взаимодействие с *объектом управления* для поддержания его функционирования в соответствии с поставленной целью. Системы управления используются в аэрокосмической отрасли, энергетике, медицине, робототехнике, массовом транспорте и др. отраслях. На каждом шаге вычислительного цикла *контроллер системы управления* получает входную информацию из окружения и обрабатывает ее. По результатам вычисления формируется управляющий сигнал для воздействия на объект управления.

Контроллеры систем управления определяют важнейший подкласс *гибридных систем*. В свою очередь, класс гибридных систем является подклассом реактивных систем в соответствии с системой классификации программ [6]. Другими подклассами являются временные,

недетерминированные и вероятностные автоматы, а также системы реального времени. Контроллер системы управления может быть составлен из частей, принадлежащим другим подклассам реактивных систем, и также являться системой реального времени или временным автоматом.

В настоящей работе рассматривается подход к разработке автоматной программы на базе ее модели (шаблона). В модели фиксирована лишь управляющая часть программы, а информационная часть схематично представлена набором произвольных данных и операторов, которые предстоит конкретизировать в автоматной программе. Модель составлена из набора *слоев* (аспектов). В простейшем случае слой может быть реализован независимой утилитой с предоставлением определенного интерфейса. В общем случае включение в модель очередного слоя требует существенной модификации исходной модели.

Далее определим базисные модели контроллеров систем управления.

Контроллеры большинства промышленных систем управления определяются следующей *простейшей моделью*. Программа контроллера реализует многократное циклическое исполнение следующего правила на языке требований:

next(in) → Step(in, s: s', com), control(com)

Срабатывание очередного шага **Step** работы контроллера инициируется приходом сообщения **next(in)**, где in – набор *входных параметров*, поступающих от *объекта управления*. Программа **Step** анализирует информацию in о текущем состоянии объекта управления и вычисляет управляющую команду **com** для модификации поведения объекта в соответствии с целями управления. Здесь s – набор переменных *состояния* системы управления; s' обозначает модификацию значения s при завершении работы **Step**. В типичном случае состояние определяется координатами объекта управления в пространстве, текущим временем и другими параметрами управления. Управляющее воздействие на объект управления реализуется посылкой сообщения **control(com)**, где **com** – команда управления объектом с целью корректировки его поведения.

В приведенной модели сообщение **next(in)** посылается *управляющей программой верхнего уровня*. Чаше реализуется другая версия данной модели:

wait T, Step(in, s: s', com), control(com)

Оператор **wait T** реализует остановку исполнения программы **Step** и ожидание в течение отрезка времени T, после чего исполнение продолжается со следующего оператора. В данной схеме не требуется управляющая программа. Время ожидания T может корректироваться программой **Step**.

Модель системы управления для конкретной прикладной области может быть построена расширением определенной выше простейшей модели набором *слоев* (аспектов), обеспечивающих ту или иную функциональность. Слой может быть реализован независимой утилитой с предоставлением определенного

интерфейса. В общем случае включение в модель очередного слоя требует существенной модификации исходной модели.

Простейшая модель системы управления может быть расширена слоями, обеспечивающими механизмы отказоустойчивости и мониторинга.

Одним из важнейших требований к функционированию системы управления является *отказоустойчивость*: возникающая аварийная (нештатная) ситуация должна автоматически распознаваться и, по возможности, преодолеваться применением механизмов восстановления нормального режима функционирования. Отказоустойчивая модель контроллера системы управления определяется ниже парой правил:

```
run:   next(in) → Step(in, s: s', com: errC #alarm), control(com)
alarm: Восстановление(s, errC : #run : err #exit)
```

Правила идентифицируются (метятся) *управляющими состояниями* *run* и *alarm*. Исполнение *гиперфункции* *Step* завершается по одной из двух *ветвей*. Исполнение, завершающееся по первой ветви, соответствует нормальному (штатному) функционированию: вычисляются *s'* и *com*, далее посылается сообщение *control(com)*, и управление возвращается в состояние *run*, где ожидается очередное сообщение *next(in)*. При обнаружении ошибочной (аварийной) ситуации гиперфункция *Step* завершается второй ветвью с выдачей кода ошибки *errC*. При этом срабатывает оператор перехода *#alarm* на исполнение второго правила, реализующего *блок восстановления*. Гиперфункция *Восстановление* пытается найти адекватное решение для парирования ошибочной ситуации. Если это удастся, исполнение гиперфункции завершается первой ветвью; оператор *#run* передает управление на исполнение первого правила. В противном случае, исполнение завершается второй ветвью; оператор *#exit* аварийно завершает исполнение программы контроллера с выдачей кода ошибки *err* и переходом в программу верхнего уровня.

Представленная компактная отказоустойчивая модель определяет гибкий поэтапный механизм восстановления при обнаружении нештатной ситуации. Восстановление работы программы контроллера в случае ее поломки реализуется другой программой, обеспечивающей общий мониторинг всей системы управления. *Мониторинг системы управления* может быть представлен следующим правилом, исполняемым циклически:

```
Monitoring(s: s', viz), show(viz)
```

Программа *Monitoring(s: s', viz)* сравнивает плановое и фактическое функционирование объекта управления, оценивает удовлетворение существующих ограничений и определяет возможность аварийных сценариев. Результат анализа процесса управления представлен набором данных *viz*. Сообщение *show(viz)* передает данные *viz* для визуализации *оператору системы*.

Мониторинг может быть реализован как внутри программы *Step*, так и независимым параллельным процессом, в частности, как часть управляющей программы верхнего уровня.

Мониторинг обычно интегрирован с другими возможностями, которые могут быть реализованы отдельными слоями. Среди них возможность переключения между *автоматическим* и *ручным* режимами управления. Решение о переходе на ручное управление принимает оператор на основе данных визуализации процесса управления. Отметим, что фиксация аварийного или катастрофического состояния должна быть упреждающей с тем, чтобы оператор имел время проанализировать угрожающую ситуацию для проведения адекватной модификации управления. Прогностический характер оценки процесса управления предполагает анализ истории функционирования объекта на определенном промежутке времени.

Заключение

В настоящей работе описывается технология автоматного программирования для разработки контроллеров систем управления на базе их моделей. Описываются две базисные модели. Представленные модели базируются на опыте работ [3, 5]. Модель управления беспилотным летальным аппаратом определена с включением трех дополнительных слоев: интеграции автоматического и ручного управления, мониторинга и защиты от несанкционированного доступа [3]. Модель космического аппарата дополнительно включает следующие слои: управление движением, поддержку работы систем энергообеспечения и терморегулирования, взаимодействие с наземным комплексом управления [5].

Отношение включения в модель очередного слоя определяет древесную структуру множества моделей автоматных программ. Представленные модели – это лишь начальный этап в дальнейшей систематической разработке древесной иерархии моделей систем управления. Данная иерархия определяет другую, ортогональную классификацию программ реактивных систем в дополнении к классификации, описанной в работе [6].

Работа выполнена при поддержке РФФИ, грант № 16-01-00498-а.

Литература

1. Шелехов В.И. Язык и технология автоматного программирования // «Программная инженерия», №4, 2014. — С. 3-15.
URL: <http://persons.iis.nsk.su/files/persons/pages/automatProg.pdf>
2. Шелехов В.И. Разработка автоматных программ на базе определения требований // Системная информатика, №4, 2014. — ИСИ СО РАН, Новосибирск. — С. 1-29.
URL: http://persons.iis.nsk.su/files/persons/pages/req_tech.pdf

3. Тумуров Э.Г., Шелехов В.И. Требования к системе управления квадрокоптером // Системная информатика. №5, 2015 — ИСИ СО РАН, Новосибирск. — С. 39-54.
URL: <http://persons.iis.nsk.su/files/persons/pages/QuadReq.pdf>
4. Тумуров Э.Г., Шелехов В.И. Технология автоматного программирования на примере программы управления лифтом // «Программная инженерия», Том 8, № 3, 2017. — С.99-111.
URL: <http://persons.iis.nsk.su/files/persons/pages/lift1.pdf>
5. Тюгашев А.А., Шелехов В.И. Модель программы управления спутником qXz // Тезисы докладов Межд. конф. «Математические методы и современные космические технологии», Алматы. Казахстан. — 2016. — С.176-179. URL: <http://persons.iis.nsk.su/files/persons/pages/ctrlspacecraft.pdf>
6. Шелехов В.И. Классификация программ, ориентированная на технологию программирования. — 2016. — ИСИ СО РАН, Новосибирск. — 11 с.
URL: <http://persons.iis.nsk.su/files/persons/pages/prog.pdf>
7. Карнаухов Н.С., Першин Д.Ю., Шелехов В.И. Язык предикатного программирования Р. — Новосибирск, 2010. — 42с. — (Препр. / ИСИ СО РАН; N 153). URL: <http://persons.iis.nsk.su/files/persons/pages/plang12.pdf>
8. Driver C., Reilly S., Linehan E., Cahill V. and Clarke S. Managing embedded systems complexity with aspect-oriented model-driven engineering // ACM Trans. Embed. Comput. Syst., Vol. 10. 2011. P. 1-26.
9. Баранов И.А. Модельно-ориентированный подход в разработке программных компонент для комплексов СМ1820М // Москва: ОАО «ИНЭУМ им. И.С. Брука». 2012. 11 с.
10. García-García JA, Ortega MA, García-Borgoñon L. NDT-Suite: a model-based suite for the application of NDT // MJ Escalona International Conference on Web Engineering. 2012. P. 469-472.
11. Шелехов В.И. Разработка и верификация алгоритмов пирамидальной сортировки в технологии предикатного программирования. — Новосибирск, 2012. — 30с. — (Препр. / ИСИ СО РАН. № 164).
12. Mittal S., Martín J. Model-driven systems engineering for netcentric system of systems with DEVS unified process // Winter Simulation Conference. 2013.
13. Schmidt, D.C. Model-Driven Engineering. IEEE Computer // Computer Society, 39. 2006. P. 25-31.
14. Семейство языков и инструментов Epsilon.
URL: <http://www.eclipse.org/epsilon>