# Модель программы управления полетом спутника qXz

А.А. Тюгашев[1], В.И. Шелехов[2]

[1] Самарский Университет им. С.П. Королева, Самара, Россия, tau797@mail.ru
[2] Институт Систем Информатики им. А.П. Ершова, Новосибирск, Россия, vshel@iis.nsk.su

Описывается модель программы управления полетом спутника Земли как часть бортовой программы управления, декомпозируемой в виде набора слоев в модельно-ориентированном подходе. Модель разрабатывается на языке требований в рамках технологии автоматного программирования. Модель обеспечивает отказоустойчивость программы управления полетом.

## Введение

Разработка программного обеспечения становится все более сложной и трудоемкой частью в общем комплексе работ при создании нового космического аппарата (КА); иногда она составляет большую часть затрат при разработке КА. Значительная часть управления КА, которая раньше реализовывалась в виде разнообразной аппаратуры, теперь становится частью программного управления. Соответственно, в связи с усложнением логики функционирования бортовой аппаратуры и необходимостью обеспечения новых требований, таких как отказоустойчивость, усложняются бортовые управляющие программы.

Совершенствование технологии бортового программирования имеет длительную историю и разнообразный опыт в космических центрах; см., например, [1]. В условиях растущей конкуренции для дальнейшего развития технологии необходимо использовать современные методы программной инженерии, в частности, формальные методы, модельно-ориентированное (model-driven engineering) и аспектно-ориентированное программирование.

В настоящей работе описывается модель бортовой программы управления спутником Земли в виде набора *слоев* (аспектов). Модель представляется в виде списка правил на *языке требований* [2], легко транслируемых в императивную программу. Исходной является простейшая модель *системы управления*, которая расширяется механизмом отказоустойчивости. Здесь используется аппарат *гиперфункций* [2], более общий и гибкий по сравнению с известным механизмом исключений языка Java.

Модель летательного аппарата [3] можно определить как расширение отказоустойчивой модели системы управления следующими слоями: интеграции автоматического и ручного управления, мониторинга и защиты от несанкционированного доступа. Модель КА дополнительно включает следующие слои: управление движением КА, поддержку работы служебных систем (энергообеспечение, терморегулирование и др.) и разнообразных сервисов, мониторинг всех процессов и подсистем КА, взаимодействие с наземным комплексом управления. 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.

*Модель процесса управления полетом* КА, представленная выше как один из слоев бортовой программы КА, включает два цикла управления: для выхода на целевую орбиту и штатного движения по орбите. Цикл управления реализуется периодическим включением *этапа коррекции* для определения текущих координат КА, оценки отклонения от целевой орбиты и вычисления параметров коррекции, определяющих изменение скорости и угловой скорости КА. При значительном отклонении управление возвращается в первый цикл, реализующий маневр КА для оптимального движения к целевой орбите. Если отклонение незначительно, реализуется корректирующее изменение ориентации КА для асимптотического приближения к целевой орбите.

# 1. Язык автоматного программирования

***Гиперфункции.*** 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.

Ветви *вызова гиперфункции* выходят в разные места программы, содержащей вызов. Вызов гиперфункции записывается в виде A(x: y #M1: z #M2). Здесь M1 и M2 – метки программы; 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: **...** может быть представлен в виде 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, 7]. Традиционные подходы в реализации обработки аварийных ситуаций предполагают заведение дополнительных структур, усложняющих программу. Этого удается избежать при использовании гиперфункций. Подробнее см. в разделе 4 работы [6], подразделе «баланс информационных и управляющих связей».

Отметим, что гиперграфовая структура автоматной программы является естественным продолжением аппарата гиперфункций.

***Автоматная программа*** определяется следующей конструкцией:

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

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.

*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 либо оператором перехода вида #M, либо нормально; в последнем случае исполнение продолжится с начала следующего сегмента. Оператор #M, где M – имя управляющего состояния, реализует переход на начало сегмента, ассоциированного с управляющим состоянием M.

Сегменты кода содержат операторы, представленные на некотором *базисном языке* программирования, предикатном [8] или императивном.

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

```
login:   if (exists(get_string()))
             #passw
         else  #login
passw:   if (valid(get_string()))
             #session
         else  #passw
session: сессия_пользователя();
         #login
```

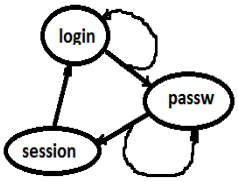


Рис. 1 – Схема работы ОС с пользователем: программа и ее автомат

В управляющем состоянии login операционная система запрашивает имя пользователя. Если полученное от пользователя имя существует в системе, она переходит в управляющее состояние passw, иначе возвращается в состояние login. В состоянии passw 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 имеет следующую структуру:

$$<\text{условие}_1>, <\text{условие}_2>,\ldots, <\text{условие}_n> \rightarrow <\text{действие}_1>, \ldots, <\text{действие}_m>;$$

*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 исполняется 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, <условие$_1$>, … , <условие$_n$> → <действие$_1$>, … , <действие$_m$>, s2

s1 → <действие$_1$>, … , <действие$_m$>, s2

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

s1: <условие$_1$>, … , <условие$_n$> → <действие$_1$>, … , <действие$_m$> #s2

s1: <действие$_1$>, … , <действие$_m$> #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. Модель системы управления

Бортовая программа управления спутником, являясь системой реального времени, принадлежит также классу *систем управления* в соответствии с системой классификации программ [5]. Типовая модель контроллера системы управления определяется многократным циклическим исполнением следующего правила на языке требований [2]:

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.

Модель системы управления для конкретной прикладной области является расширением определенной выше модели, которая дополняется набором *слоев* (аспектов), обеспечивающих ту или иную функциональность. На примере программы управления квадрокоптером определена типовая модель управления беспилотным летальным аппаратом с включением трех дополнительных слоев: интеграции автоматического и ручного управления, мониторинга и защиты от несанкционированного доступа [3]. 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.

Одним из важнейших требований к функционированию подсистем спутника 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. *Гиперфункция* Step имеет две *ветви*. Исполнение гиперфункции 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 удается, 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.

## 3. Структура бортовой программы спутника

Проведение исследований в космическом пространстве, обеспечение инфраструктуры точного позиционирования системой ГЛОНАСС, космическое картографирование, видеонаблюдение за поверхностью Земли и многое другое составляет *целевую задачу*, реализуемую космическим аппаратом. Для ее успешного выполнения следует обеспечить необходимую инфраструктуру – *окружение* для реализации целевой задачи. Ниже представлена модель бортовой программы спутника qXz.

```
process _qXz {
        ОсновнаяМиссия ||
        УправлениеПолетом ||
        Функционирование ||
        Мониторинг ||
        УправлениесЗемли
}
```

Приведенная выше модель определяет параллельную композицию пяти процессов. Главный процесс ОсновнаяМиссия реализует программное управление для выполнения целевой задачи аппарата qXz. Процесс УправлениеПолетом реализует управление движением спутника. Процесс Функционирование определяет поддержку работы служебных систем (энергообеспечение, терморегулирование и др.) и разнообразных сервисов; каждая из подсистем функционирует как независимый параллельный процесс. Процесс Мониторинг отслеживает состояние всех перечисленных процессов и подсистем спутника. Процесс УправлениесЗемли исполняет команды оператора с наземного комплекса управления (НКУ) спутником.

Каждая подсистема из набора процессов Функционирования определена в виде *интерфейса*, содержащего набор объектов и операций, доступных извне подсистемы. Жизнеобеспечение аппаратуры, принадлежащей процессу ОсновнаяМиссия и монтирующейся внутри спутника, реализуется через интерфейсные объекты подсистем Функционирования.

Автономный контроль работоспособности реализуется каждой подпрограммой в составе программы спутника. Глобальный контроль реализуется независимым процессом Мониторинг, который отслеживает состояние всех процессов в рамках бортовой программы спутника, а также подсистем спутника, обнаруживает аварийные ситуации и производит по ним соответствующие управляющие воздействия для восстановления нормальной работы. Функцией процесса Мониторинг является также накопление истории работы подсистем спутника и взаимодействие с наземным комплексом управления (НКУ) спутником. В каждом независимом процессе осуществляется локальный мониторинг.

## 4. Модель процесса управления полетом

**Состояние** процесса УправлениеПолетом.

```
S s; // состояние спутника: время, координата центра масс спутника в инерционной системе
        координат, направление полета, скорость и угловая скорость спутника на момент
        последнего сеанса навигации; текущие режимы работы бортовой аппаратуры; в
        начальный момент не определено. Здесь s – переменная, значение которой определяет
        состояние спутника, S – тип переменной s, который будет определен позже.
time Tw; // время ожидания следующего сеанса навигации
```

**Управляющие состояния**:

start – начало очередного цикла ориентации спутника;
out – спутник пока вне орбиты – режим движения спутника по направлению к орбите;
run – штатный режим движения спутника по орбите;
alarm – режим восстановления после аварийной ситуации.

**Локальные программы.**

Ориентация( : s #out : s #run : errC #alarm) – определяет координаты спутника в инерциальной системе координат, направление полета, скорость и угловую скорость. Их значения формируются параметром-результатом s. Если отклонение спутника от орбиты незначительное, реализуется выход run, иначе out. В случае, когда проведение ориентации спутника невозможно, реализуется выход alarm с выдачей кода ошибки errC.

Маневр(s : s′, Tw : errC #alarm) – реализация маневра для выхода на целевую орбиту. Новое значение состояния спутника s′ учитывает изменения высоты и наклонения орбиты. Определяется также время Tw следующего сеанса ориентации спутника. В случае невозможности проведения маневра из-за отказа аппаратуры, реализуется выход alarm с кодом ошибки errC.

Коррекция(s : s′, Tw : errC #alarm) – реализуется коррекция незначительного отклонения от целевой орбиты.

Восстановление(s, errC : #start : err #exit) – на основе анализа текущего состояния спутника s и кода ошибки errC реализуется процесс восстановления штатного режима полета спутника. В ситуации, когда восстановление невозможно, реализуется выход exit с кодом ошибки err, аварийно завершающий исполнение программы УправлениеПолетом.

**Требования.**

```
process УправлениеПолетом( : err #exit) {
    start:  Ориентация(: s #out: s #run: errC #alarm);
    out:    Маневр(s : s′, Tw : errC #alarm), wait Tw      #start
    run:    Коррекция(s : s′, Tw : errC #alarm), wait Tw   #start
    alarm:  Восстановление(s, errC : #start : err #exit)
}
```

При запуске программы состояние спутника s неопределенно. Цель программы – выведение спутника на целевую орбиту с дальнейшим неограниченным по времени полетом по орбите. Очередной цикл работы программы начинается с процедуры ориентации спутника, в результате чего определяются его координаты, направление полета, скорость и угловая скорость. Далее оценивается величина отклонения спутника от целевой орбиты. Если отклонение значительное, запускается программа Маневр для реализации оптимального режима приближения к целевой орбите. Рассчитываемое время Tw следующего сеанса ориентации может уменьшаться при приближении к целевой орбите. В случае аварийной ситуации запускается программа Восстановление, которая пытается устранить причину аварии, и если это удается, возобновляется нормальную работу программы, иначе реализуется аварийное завершение с выдачей кода ошибки err.

Определение текущего положения КА на орбите производится с применением нижеследующих средств. Приборный состав охватывает звёздные датчики, аналоговые магнитометры, одноосные измерители угловой скорости, оптический солнечный датчик. Управляющие моменты, прилагаемые к корпусу КА по оси связанной системы координат в процессе управления угловым движением космического аппарата, создаются с помощью управляющих двигателей маховиков и силовых магнитов [4].

Алгоритм коррекции движения КА предназначен для расчёта параметров коррекции, определяющих требуемое изменение скорости и угловой скорости после проведения

очередного этапа коррекции. Также вычисляется момент времени, когда необходимо применить следующий этап коррекции. Расчёт исходных данных коррекции состоит в разложении пространственного движения на более простые (плоские) движения, суперпозиция которых даёт траекторию переориентации. Коррекция, как угловое перемещение представляется в виде совокупности вращений трёх опорных координатных базисов. Для заданного момента времени алгоритм вычисляет требуемые значения вектора скорости и кватерниона, определяющих движение КА. Траектория на интервале между двумя соседними коррекциями строится на основе суммирования движений, соответствующих гашению скорости, позиционному переходу, набору скорости. Изменение модуля скорости при коррекции в общем случае характеризуют четыре участка: выход на максимальное значение модуля скорости, движение с постоянной скоростью, торможения и движения с заданной скоростью.

*Работа выполнена при поддержке РФФИ, грант № 12-01-00686.*

## Список литературы

1. Колташев А.А., Кочура С.Г. Технология создания и сопровождения бортового программного обеспечения спутников связи, навигации и геодезии: современное состояние // Наукоемкие технологии, № 9, т. 15, 2014. — С. 39-42.
2. Шелехов В.И. Разработка автоматных программ на базе определения требований // Системная информатика, №4, 2014. — ИСИ СО РАН, Новосибирск. — С. 1-29. 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. Салмин В.В., Филатов А.В., Ткаченко И.С., Тюгашев А.А., Сопченко Е.В. Вычислительный алгоритм формирования программного движения в программном повороте малого космического аппарата // Вестник Самарского государственного аэрокосмического университета им. академика С.П. Королёва (национального исследовательского университета). 2015. Т. 14. № 2. — С. 9-19.
5. Шелехов В.И. Классификация программ, ориентированная на технологию программирования. — 2016. — ИСИ СО РАН, Новосибирск. — 11с. http://persons.iis.nsk.su/files/persons/pages/req_tech.pdf
6. Тумуров Э.Г., Шелехов В.И. Технология автоматного программирования на примере программы управления лифтом. — ИСИ СО РАН, Новосибирск, 2016. – 18с. URL: http://persons.iis.nsk.su/files/persons/pages/lift1.pdf
7. Шелехов В.И. Разработка и верификация алгоритмов пирамидальной сортировки в технологии предикатного программирования. – Новосибирск, 2012. – 30с. – (Препр. / ИСИ СО РАН. № 164).
8. Карнаухов Н.С., Першин Д.Ю., Шелехов В.И. Язык предикатного программирования Р. – Новосибирск, 2010. – 42с. – (Препр. / ИСИ СО РАН; N 153).