УДК 681.32

О. А. Антамошкин, Ю. А. Нургалеева, А. В. Усачев

МОДЕЛИ МОДУЛЬНОЙ ДЕКОМПОЗИЦИИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ТЕХНОЛОГИЧЕСКИХ ЦИКЛОВ УПРАВЛЕНИЯ

Рассмотрены модели модульной декомпозиции программного обеспечения, реализующего технологические циклы управления.

Ключевые слова: программное обеспечение, технологический цикл управления, модульная декомпозиция.

Разработка и эксплуатация современных космических аппаратов (КА), на базе которых создаются распределенные системы реального времени, непосредственно связана с разработкой программного обеспечения (ПО), функционирующего как в контурах наземного комплекса управления, так и в бортовых комплексах управления, выполненных на основе бортовых цифровых вычислительных машин. В настоящее время в разработке и эксплуатации КА используется сочетание серийных методов с индивидуальным производством, которое, с одной стороны, характеризуется высокой степенью унификации базового ПО, инженерных методов и методик его проектирования, а с другой – применением уникального ПО (даже для КА одного типа).

Процесс эксплуатации и функционирования КА на орбите регламентируется технологическими циклами управления (ТЦУ), которые определяют временные и информационные взаимосвязи между отдельными контурами управления КА, что выражается в установлении периодичности, планируемой длительности и порядка решения задач по обработке информации и управления КА [1]. В процессе эксплуатации возможно изменение характеристик КА и режимов его функционирования вследствие технических неисправностей или изменений в программах целевой работы, отражающихся как на составе, так и на алгоритмах управляющего ПО. При этом в случае выведения КА из нештатных состояний (в соответствии с нештатными ТЦУ) ряд характеристик управляющего ПО, а также процесса его разработки, в частности надежность, отказоустойчивость, быстродействие, определяет вероятность успешного завершения этой операции и размер ущерба [2].

На современном этапе среди множества требований к качеству ПО, методам и методикам его разработки и сопровождения большое значение приобретают такие требования, как скорость и стоимостная эффективность разработки, внешние средства поддержки исполнения, тестирования и анализа, а также доступность методов и инструментальных средств широкому кругу специалистов, обслуживающих системы управления КА и ТЦУ [3–6].

Выполнение этих требований при проектировании отказоустойчивых и высоконадежных систем ПО связано с созданием моделей модульной декомпозиции при условии их интеграции с методами инженерного проектирования ПО и адекватного отражения технологии управления КА, что позволяет использовать многолетний опыт и интуицию проектировщиков систем управления КА, ответственных за его функционирование на орбите и выполнение целевых функций [7].

Основные требования к ПО и типовая структура модульного ПО ТЦУ определяются спецификой функционирования системы управления КА в реальных условиях и введением ПО процессов управления КА в ранг программного продукта. Среди этих требований в первую очередь выделяются необходимость эффективного использования памяти и производительности; способность к массовому тиражированию, длительность и непрерывность функционирования; высокая степень документированности; эффективность эксплуатации, модернизируемость; переносимость и сопровождаемость [8]. Все перечисленные требования должны выполняться на фоне реализации основных функциональных задач космической системы.

Известны два типа моделей модульной декомпозиции программного обеспечения вычислительных и информационно-управляющих систем [9]:

– модель потока данных;

– модель объектов.

В основе модели потока данных лежит разбиение по функциям. Модель объектов основана на слабо сцепленных сущностях, имеющих собственные наборы данных, состояния и наборы операций.

Очевидно, что выбор типа декомпозиции ПО технологических циклов управления должен определяться сложностью структуры ТЦУ, количеством контуров и подсистем, программная составляющая которых подвергается декомпозиции.

Итак, модуль – это фрагмент программного текста, являющийся строительным блоком для физической структуры системы. Как правило, модуль состоит из интерфейсной части и части-реализации.

Модульность является свойством системы, которая может подвергаться декомпозиции на ряд внутренне связанных и слабо зависящих друг от друга модулей.

По определению Г. Майерса, модульность – это свойство ПО, обеспечивающее интеллектуальную возможность создания сколь угодно сложной программы [6]. Проиллюстрируем эту точку зрения.

Пусть $C(x)$ – функция сложности решения проблемы x, $T(x)$ – функция затрат времени на решение проблемы x. Для двух проблем p_1 и p_2 из соотношения $C(p_1) > C(p_2)$ вытекает, что

$$T(p_1) > T(p_2). \quad (1)$$

Этот вывод интуитивно ясен: решение сложной проблемы требует большего времени.

Далее отметим, что из практики решения проблем человеком следует

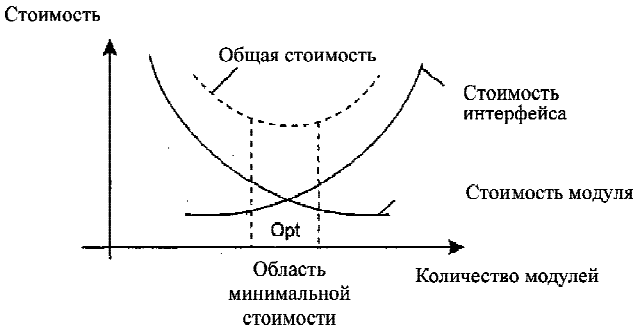
$$C(p_1 + p_2) > C(p_1) + C(p_2),$$

отсюда, с учетом соотношения (1), запишем

$$T(p_1 + p_2) > T(p_1) + T(p_2). \quad (2)$$

Соотношение (2) – это обоснование модульности. Оно приводит к заключению, что сложную проблему легче решить, разделив ее на управляемые части. Результат, выраженный неравенством (2), имеет важное значение и для модульности ПО. Фактически это аргумент в ее пользу.

Однако здесь отражена лишь часть реальности, поскольку тут не учитываются затраты на межмодульный интерфейс, что является существенным фактором при проектировании ПО ТЦУ: увеличение количества модулей (и уменьшение их размера) повышает эти затраты (см. рисунок).



Затраты на модульность

Таким образом, существует оптимальное количество модулей Opt, которое приводит к минимальной стоимости разработки, однако не существует метода гарантированного предсказания Opt. Впрочем разработчики знают, что оптимальный модуль должен удовлетворять двум критериям [9]:

– снаружи он проще, чем внутри;

– его проще использовать, чем построить.

Принцип информационной закрытости Д. Парнаса (1972) утверждает: содержание модулей должно быть скрыто друг от друга [7; 8]. Модуль должен определяться и проектироваться так, чтобы его содержимое (процедуры и данные) было недоступно тем модулям, которые не нуждаются в такой информации, т. е. клиентам.

Информационная закрытость означает следующее:

– все модули независимы и обмениваются только информацией, необходимой для работы;

– доступ к операциям и структурам данных модуля ограничен.

Отметим достоинства информационной закрытости:

– возможна разработка модулей различными независимыми коллективами;

– обеспечивается легкая модификация системы за счет того, что вероятность распространения ошибок очень мала, поскольку большинство данных и процедур скрыто от других частей системы.

Идеальный модуль играет роль «черного ящика», содержимое которого невидимо клиентам. Он прост в использовании, так как количество ручек и органов управления им невелико [9]. Его легко развивать и корректировать в процессе сопровождения программной системы. Но для обеспечения таких возможностей система внутренних и внешних связей модуля должна отвечать особым требованиям. Обсудим характеристики внутренних и внешних связей модуля.

Связность модуля (*Cohesion*) – это мера зависимости его частей [7–9]. Она является внутренней характеристикой модуля. Чем выше связность модуля, тем лучше результат проектирования, т. е. тем «чернее» его ящик (капсула, защитная оболочка модуля), тем меньше ручек управления на нем находится и тем проще эти ручки.

Для измерения связности используют понятие силы связности (СС). Существует семь типов связности (см. таблицу):

1) связность по совпадению (СС = 0): в модуле отсутствуют явно выраженные внутренние связи;

2) логическая связность (СС = 1): части модуля объединены по принципу функционального подобия. Например, модуль состоит из разных подпрограмм обработки ошибок. При использовании такого модуля клиент выбирает только одну из подпрограмм. Недостатки логической связности: сложное сопряжение; большая вероятность внесения ошибок при изменении сопряжения ради одной из функций;

3) временная связность (СС = 3): части модуля не связаны, но необходимы в один и тот же период работы системы. Недостаток – сильная взаимная связь с другими модулями, отсюда сильная чувствительность к внесению изменений;

4) процедурная связность (СС = 5): части модуля связаны порядком выполняемых ими действий, реализующих некоторый сценарий поведения;

5) коммуникативная связность (СС = 7): части модуля связаны по данным, т. е. работают с одной и той же структурой данных);

6) информационная (последовательная) связность (СС = 9): выходные данные одной части используются как входные данные в другой части модуля;

Характеристика связности модуля

Тип связности	Сопровождаемость	Роль модуля
Функциональная	Лучшая сопровождаемость	«Черный ящик»
Информационная (последовательная)		Не совсем «черный ящик»
Коммуникативная		«Серый ящик»
Процедурная	Худшая сопровождаемость	«Белый» или «просвечивающий ящик»
Временная		«Белый ящик»
Логическая		
По совпадению		

7) функциональная связность (СС = 10): части модуля вместе реализуют одну функцию.

Отметим, что типы связности 1, 2, 3 – это результат неправильного планирования архитектуры, а тип связности 4 – результат небрежного планирования архитектуры приложения [2].

Функционально связный модуль содержит элементы, участвующие в выполнении одной и только одной проблемной задачи, – функции ТЦУ. Каждый из этих модулей имеет единичное назначение. Когда клиент вызывает модуль, выполняется только одна работа, без привлечения внешних обработчиков. Некоторые из функционально связных модулей очень просты (например, «Вычислять синус угла» или «Читать запись файла»), другие сложны (например, «Вычислять координаты цели»). Модуль «Вычислять синус угла», очевидно, реализует единичную функцию. Дело в том, что – несмотря на сложность модуля и на то, что его обязанность исполнять несколько подфункций, если его действия можно представить как единую проблемную функцию (с точки зрения клиента), все равно считается, что модуль функционально связен.

Приложения, построенные из функционально связных модулей, легче всего сопровождать. Можно даже подумать, что любой модуль можно рассматривать как однофункциональный, однако не стоит заблуждаться. Существует много разновидностей модулей, которые выполняют для клиентов перечень различных работ и этот перечень нельзя рассматривать как единую проблемную функцию. Критерий при определении уровня связности этих нефункциональных модулей таков: как связаны друг с другом различные действия, которые они исполняют.

Приведем алгоритм определения уровня связности модуля.

1. Если модуль – единичная проблемно-ориентированная функция, то уровень связности – функциональный; конец алгоритма. В противном случае перейти к п. 2.

2. Если действия внутри модуля связаны, то перейти к п. 3. Если действия внутри модуля никак не связаны, то перейти к п. 6.

3. Если действия внутри модуля связаны данными, то перейти к п. 4. Если действия внутри модуля связаны потоком управления, то перейти к п. 5.

4. Если порядок действий внутри модуля важен, то уровень связности – информационный. В противном случае уровень связности – коммуникативный. Конец алгоритма.

5. Если порядок действий внутри модуля важен, то уровень связности – процедурный. В противном случае уровень связности – временной. Конец алгоритма.

6. Если действия внутри модуля принадлежат к одной категории, то уровень связности – логический. Если действия внутри модуля не принадлежат к одной категории, то уровень связности – по совпадению. Конец алгоритма.

Возможны более сложные случаи, когда с модулем ассоциируются несколько уровней связности. В этих случаях следует применять одно из двух правил:

– правило параллельной цепи: если все действия модуля имеют несколько уровней связности, то модулю присваивается самый сильный уровень связности;

– правило последовательной цепи: если действия в модуле имеют разные уровни связности, то модулю присваивают самый слабый уровень связности.

Например, модуль может содержать некоторые действия, которые связаны процедурно, а также другие действия, связные по совпадению. В этом случае применяют правило последовательной цепи и модуль в целом считают связным по совпадению.

Сцепление (*Coupling*) – мера взаимозависимости модулей по данным [5; 6]. Это внешняя характеристика модуля, которую желательно уменьшать.

Количественно сцепление измеряется степенью сцепления (СЦ). Выделяют шесть типов сцепления:

1) сцепление по данным (СЦ = 1): модуль А вызывает модуль В. Все входные и выходные параметры вызываемого модуля – простые элементы данных;

2) сцепление по образцу (СЦ = 3): в качестве параметров используются структуры данных;

3) сцепление по управлению (СЦ = 4): модуль А явно управляет функционированием модуля В (с помощью флагов или переключателей), посылая ему управляющие данные;

4) сцепление по внешним ссылкам (СЦ = 5): модули А и В ссылаются на один и тот же глобальный элемент данных;

5) сцепление по общей области (СЦ = 7): модули разделяют одну и ту же глобальную структуру данных;

6) сцепление по содержанию (СЦ = 9): один модуль прямо ссылается на содержание другого модуля (не через его точку входа), например коды их команд перемежаются друг с другом.

В простейшем случае сложность системы определяется как сумма мер сложности ее модулей. Сложность модуля может вычисляться различными способами. Вместе с тем известно, что любая сложная система состоит из элементов и системы связей между элементами и что игнорировать внутрисистемные связи неразумно.

При оценке сложности ПО Т. МакКейб предложил исходить из топологии внутренних связей [4]. Для этой цели он разработал метрику цикломатической сложности. Это был шаг в нужном направлении. И дальнейшее уточнение оценок сложности потребовало, чтобы каждый модуль мог представляться как локальная структура, состоящая из элементов и связей между ними.

Таким образом, при комплексной оценке сложности ПО ТЦУ необходимо рассматривать меру сложности модулей, меру сложности внешних связей (между модулями) и меру сложности внутренних связей (внутри модулей) [2; 5]. Традиционно с внешними связями сопоставляют такую характеристику, как сцепление, а с внутренними связями – связность.

Библиографический список

1. Ковалев, И. В. Стохастическая сетевая модель для оптимизации формирования технологических циклов управления космическими аппаратами / И. В. Ковалев, А. А. Ступина // Авиакосмич. приборостроение. 2006. № 5. С. 2–15.

2. Ковалев, И. В. Архитектурная надежность программного обеспечения автоматизированной системы техни-

ческой диагностики / И. В. Ковалев, А. А. Ступина, Р. Ю. Царев // Приборы и системы. Управление, контроль, диагностика. 2006. № 12. С. 47–52.

3. Боэм, Б. Характеристики качества программного обеспечения / Б. Боэм, Дж. Браун, Х. Каспар и др. М. : Мир, 1999.

4. Глас, Р. Руководство по надежному программированию : пер. с англ. / Р. Гласс. М. : Финансы и статистика, 2000.

5. Дилон, Б. Инженерные методы обеспечения надежности систем / Б. Дилон, И. Сингх. М. : Мир, 1992.

6. Майерс, Г. Надежность программного обеспечения : пер. с англ. / Г. Майерс ; под ред. В. Ш. Кауфмана. М. : Мир, 1998.

7. Мамиконов, А. Г. Типизация разработки модульных систем обработки данных / А. Г. Мамиконов, В. В. Кульба, С. А. Косяченко. М. : Наука, 1989.

8. Мамиконов, А. Г. Синтез оптимальных модульных систем обработки данных / А. Г. Мамиконов, В. В. Кульба. М. : Наука, 1986.

9. Орлов, С. А. Технологии разработки программного обеспечения / С. А. Орлов. СПб. : Питер, 2002.

O. A. Antamoshkin, Ju. A. Nurgaleeva, A. V. Usachev

MODULAR DECOMPOSITION MODELS OF TECHNOLOGICAL CONTROL CYCLES SOFTWARE

Modular decomposition models of technological control cycles software are considered.

Keywords: software, technological control cycles, modular decomposition.

УДК 681.3

П. В. Зеленков, М. А. Селиванова, В. В. Брезицкая, А. П. Хохлов

МОДУЛЬ ОБРАБОТКИ ИНФОРМАЦИОННЫХ ЗАПРОСОВ ПОЛЬЗОВАТЕЛЕЙ В СЕТЬ ИНТЕРНЕТ ДЛЯ КОРПОРАТИВНЫХ ИНФОРМАЦИОННО-УПРАВЛЯЮЩИХ СИСТЕМ

Предложена новая технология организации запросов пользователей в сеть Интернет, применяемая в корпоративных информационно-управляющих системах. Предложена структура модуля обработки запросов пользователя, состоящая из агентов: получения запросов, распознавания профиля пользователя, распознавания запроса, сравнения типовых запросов, поиска информации и отображения отклика системы. Применение предлагаемой технологии позволяет существенно снизить внешний интернет-трафик.

Ключевые слова: корпоративные информационно-управляющие системы, Интернет, пользователь сети, структура системы, пользовательский запрос.

В настоящее время большинство компаний для информационного обеспечения сотрудников при взаимодействии с внешними информационными ресурсами используют подключение к глобальной сети Интернет. Для этого применяется наиболее популярная схема подключения (рис. 1). Данная схема является очень удобной, что связано с возможностью контролировать действия сотрудников в глобальной сети и закрывать доступ к сомнительным сайтам, таким как набирающие популярность социальные сети, развлекательные порталы и т. п.

В классической схеме подключения (см. рис. 1) пользователь может напрямую обратиться к Proxy-серверу и работать в сети Интернет без использования корпоративной информационно-управляющей системы (КИУС). Это приводит к тому, что запросы пользователя, связанные с рабочим процессом, никак не учитываются в рамках этой системы. Если рассмотреть работу сотрудников одного отдела в рамках предприятия, то можно увидеть, что они, как правило, работают с одной и той же информацией. Кроме того, большинство сотрудников при входе в сеть Интернет просматривают одни и те же информационные и новостные ресурсы. Следовательно, интернет-трафик, затрачиваемый на просмотр данных ресурсов, расходуется нерационально: он увеличивается соответственно числу сотрудников или более, если один сотрудник просматривает данный ресурс неоднократно.

Необходимо отметить, что в небольших компаниях выход в сеть Интернет может быть организован вообще напрямую, т. е. без использования Proxy-сервера. Это влечет за собой еще более нерациональное использование интернет-ресурсов и увеличение затрат рабочего времени сотрудников, не связанных с выполнением рабочих процессов.