электронный научно-технический журнал
ИНЖЕНЕРНЫЙ ВЕСТНИК
Издатель ФГБОУ ВПО "МГТУ им. Н.Э. Баумана". Эл No. ФС77-51036. ISSN 2307-0595

Принципы проектирования встраиваемых систем на основе программных средств

 # 11, ноябрь 2014

Кондрашов К. С., Жураковский В. Н., Силин С. И.

УДК: 303.732.4

Россия, МГТУ им. Н.Э. Баумана
kondrashovks1989@gmail.com

Введение

В наши дни разнообразные технические задачи решаются при помощи встраиваемых систем, которые представляют собой совокупность программно-аппаратных средств. В них существенную роль играет программное обеспечение, которое зачастую, особенно в интеллектуальных системах, имеет функциональную нагрузку даже большую, чем аппаратные ресурсы. Это обусловлено гибкостью изменения таких программ по сравнению с аппаратурой и возможностью существенно расширить возможности технической системы за счет чисто программных решений. При этом программное обеспечение встраиваемых систем имеет ряд существенных особенностей, отличающих его от программ общего назначения.

В статье раскрываются особенности проектирования программного обеспечения встраиваемых систем, приводятся принципы применения системного подхода к его разработке, а также описываются программные решения, которые успешно используются в ряде существующих программных продуктов во встраиваемых системах.

1. Постановка задачи

Рассмотрим основные сложности, с которыми сопряжена разработка встраиваемых программных систем [1]:

1. Такие системы имеют существенную функциональную нагрузку, которая обуславливает их сложность. Отладка программного обеспечения при этом представляет собой непростую задачу, поскольку сопряжена с необходимостью учитывать взаимодействие всех компонентов системы.
2. Зачастую доступ к встраиваемой системе в рабочих условиях получить сложно, особенно это касается авиационных, космических и других, работающих в особых условиях окружающей среды. При этом важно отметить высокие требования к надежности и корректности их работы, особенно в нештатных ситуациях.

3. В процессе отладки и тестирования бывает непросто имитировать внешнее окружение (в том числе аппаратное), что обусловлено специфическим применением встраиваемых систем.
4. Натурный эксперимент, который требуется организовать для полноценной проверки программных систем, как правило, имеет немалую стоимость или вообще нереализуем.

2. Системный подход к разработке программного обеспечения

Значительная функциональная нагрузка на встраиваемую систему, в соответствии с системным подходом [2], предполагает декомпозицию структуры системы на составляющие с выделением подсистем. Свойства системы в целом при этом определяются свойствами подсистем, из которых она состоит и связей между ними.

Под подсистемами программного обеспечения понимаются программные модули, реализующие ту или иную функциональность. Процесс разбиения на модули является иерархическим, при котором подсистемы продолжают делиться на подсистемы более низкого уровня до тех пор, пока следующий шаг разбиения не приводит к усложнению описания системы. При этом следует руководствоваться принципом единственной ответственности, принятом в объектно-ориентированном проектировании [3]: каждый модуль можно описать так, чтобы он реализовывал свою уникальную функциональность, не перекрывающуюся с функциональностью других модулей. Число подмодулей каждого модуля должно быть как можно меньше, но так, чтобы при этом подмодули не были «перегружены» с точки зрения функциональности.

Например, рассмотрим многоканальную систему сопровождения с широкой номенклатурой целей (см. рис. 1). В ней предусмотрены несколько трактов сопровождения: радиолокационный, оптический и так далее. Предполагается, что программное обеспечение для микропроцессора осуществляет обработку информации в различных трактах в реальном времени, принятие решений о приоритетах целей, рациональное распределение аппаратных ресурсов и воздействие на объекты управления.

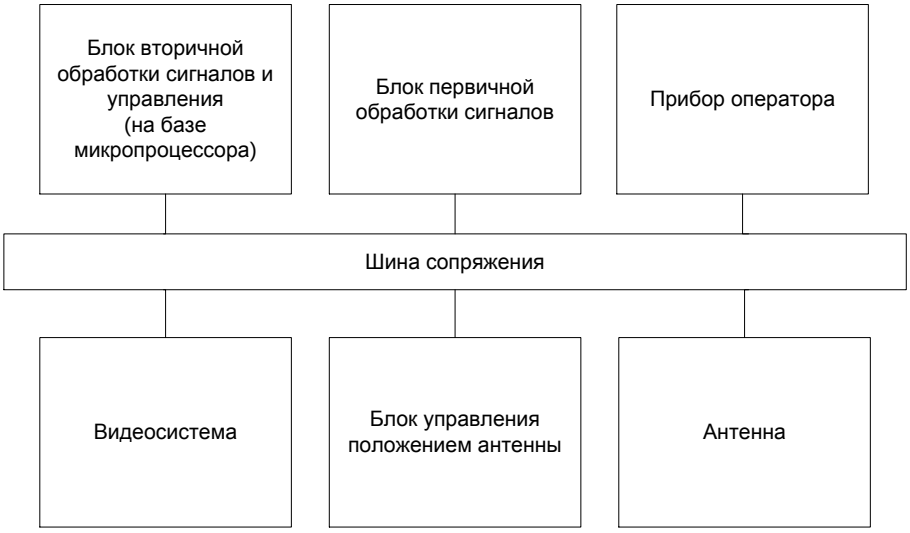

Рис. 1. Структура системы сопровождения целей.

В соответствии с высказанными соображениям, программное обеспечение может делиться на следующие модули:

1. Траекторной обработки;
2. Классификации целей по степени опасности;
3. Формирования управляющих воздействий;
4. Предварительной обработки радиолокационной и видеоинформации;
5. Распределения аппаратных ресурсов;
6. Сопряжения с внешними системами.

Данное разбиение соответствует первому иерархическому уровню и предназначено для решения одной из глобальных задач, стоящих перед системой.

В свою очередь, например, модуль формирования управляющих воздействий состоит из модулей следующего уровня (второго):

1. Экстраполятора положения целей на момент выдачи управляющего воздействия с учетом инерционности органов управления;
2. Формирователя воздействий в соответствии с заданным форматом, воспринимаемом аппаратурой;
3. Формирователя управляющих воздействий для трактов приема информации.

Аналогичным образом можно выделить модули второго уровня в остальных модулях первого, а также модули третьего уровня в модулях второго. Разбиение можно продолжать до выделения базовых математических функций или низкоуровневых запросов к аппаратным ресурсам, после чего описание программной системы неоправданно усложняется.

Если миновать каждый из иерархических уровней, и перейти сразу к нижнему, то описание программной системы становится громоздким, плохо воспринимаемым и практически не изменяемым вследствие возникновения большого числа дополнительных связей между компонентами.

Программное обеспечение является подсистемой в системе сопровождения целей, которая, в свою очередь, входит в глобальную систему взаимодействия («станция – цель»). У глобальной системы должны (в соответствии с техническим заданием) присутствовать свойства (в частности, возможность сопровождать цели с определенными характеристиками), которые накладывают зачастую сложно формализуемые ограничения и требования к свойствам подсистем, а именно на систему сопровождения и как следствие, ее программное обеспечение. В этой связи, разработку и отладку программных средств нельзя осуществлять в отрыве от глобальной системы взаимодействия. Поэтому к ней тоже целесообразно применить методы системного анализа для декомпозиции, выделения системообразующих свойств и связей, которые в большей степени определяют требования к программным средствам.

Например, при описании свойств глобальной системы нецелесообразно подробно описывать передачу сигналов в среде распространения в интересах предъявления требований к программному обеспечению. Важным моментом является лишь результат физических процессов, протекающих в глобальной системе, с точки зрения их динамических свойств. Это позволяет при отработке программных средств заменить глобальную систе-

му взаимодействия на ее модель, которая определяется свойствами, выделенными на этапе системного анализа.

3. Технология разработки программного обеспечения

В связи с необходимостью разработки программных комплексов для встраиваемых систем в условиях, приведенных в предыдущем пункте, целесообразным представляется максимально возможная отработка комплекса в среде имитационного моделирования в виде математических моделей, либо в составе комплекса при проведении пусконаладочных работ. Данный подход позволяет существенно сократить цикл разработки и затраты на отладку, настройку и осуществление экспериментальных исследований системы.

Применение системного подхода к декомпозиции программного обеспечения позволяет разбить его на модули таким образом, чтобы существовала возможность отладки отдельно взятого модуля с использованием программного имитатора, реализующего внешнюю связь данного модуля с внешней средой и остальными модулями программного обеспечения.

В зависимости от иерархического уровня модуля и желаемого уровня детализации его окружения, сложность имитатора внешней обстановки может варьироваться. Целесообразным является использование моделей разной степени детализации на разных этапах процесса разработки.

На начальном этапе разработки программного обеспечения, при проектировании модулей низкого иерархического уровня, стоит использовать так называемое модульное тестирование, предусматривающее проверку соответствия входа и выхода при условии нахождения входа в заданном диапазоне, а также отработку некорректных входных условий.

Модульное тестирование позволяет проверить функциональность модуля с точки зрения корректности написания текста программы, и его соответствие теоретической задумке разработчика. Но для проверки временных ограничений модульного теста в общем случае недостаточно. При возрастании сложности внутренней структуры модуля, а следовательно возрастании сложности входных воздействий для его проверки (такие воздействия представляют собой функции времени), соответствующее описание модульного теста может оказаться слишком громоздким, либо вообще невозможным.

В связи с этим для отладки модулей более высокого уровня используется модельное окружение имитированного и реального масштабов времени.

Моделирование в имитированном масштабе времени предполагает помещение модуля или группы модулей в среду имитационного моделирования с внутренним диспетчером, имеющим собственную шкалу времени. События в системе рассчитываются заранее и для соответствующих событий запускаются код модулей, который поставлен в соответствие этим событиям.

Такое моделирование позволяет проверить работоспособность модулей, и реализованных в них алгоритмов при задании входных воздействий сложной формы, а также «проигрывания» различных сценариев входных воздействий.

Такая имитационная модель может содержать описание окружающей среды сколь угодно большей степени детализации без учета вычислительной сложности при использовании соответствующей платформы. То есть существует возможность полностью сымитировать глобальную систему в соответствии с принципами, указанными выше. При этом выходные данные соответствуют использованию встраиваемой вычислительной аппаратуры бесконечной производительности.

Следующим этапом является перенос модели в среду реального или псевдореального масштаба времени. При этом используется аппаратная платформа с удобным доступом к отладочным интерфейсам и с возможностью записи результатов вычислений.

В связи с необходимостью работы в реальном времени [4] сложность моделей окружения, как правило, уменьшается, и модели заменяются своими эквивалентами с низкой сложностью вычислений.

На данном этапе проверяются временные ограничения и возможность функционирования выбранной реализации в заданных временных рамках.

Следующим этапом является реализация программного обеспечения на базе встраиваемой аппаратной платформы. Так как зачастую имитация соответствующих входных воздействий в составе отдельного блока или комплекса является затруднительной, то необходимо предусмотреть возможность проведения полунатурного моделирования в составе программно-аппаратного комплекса или стенда. Это достигается посредством формирования дополнительных сообщений в протоколах информационных обменов по штатным интерфейсам. Наряду с применением отладочных интерфейсов, это позволяет отлаживать модули программного обеспечения в реальном времени. Однако, отладочные интерфейсы отсутствуют в штатной работе, а дополнительные сообщения в штатных обменах позволяют сохранять информацию, которая может понадобиться на этапе испытаний и эксплуатации, что особенно важно для автономных систем.

Отметим тот факт, что описанные этапы разработки предполагают явное разделение предметно-ориентированного кода, связанного с решаемой в конкретном модуле задачей, и сервисного кода [5]. Организация программы таким образом, чтобы эти две зоны ответственности не перекрывались, является первостепенной задачей программиста. «Смешение» логики программы и обменов данными приводит к плохо читаемой программе и противоречит методам системного анализа, используемых при формировании ее структуры.

4. Пример архитектуры встраиваемой системы

Основой для создания программного обеспечения встраиваемых систем являются системные примитивы, используемые для обмена данными и поддержания реального масштаба времени выполнения тех или иных операций (в частности, обеспечения их параллельности).

В основе программного обеспечения встраиваемой системы, функционирующей на базе микропроцессора, лежит системная библиотека, которую удобно разделить на две: платформенно-зависимую и платформенно-независимую. Последняя использует первую для обращения к аппаратным ресурсам. Предметно-ориентированное программное обеспечение, в свою очередь, использует платформенно-независимую библиотеку.

Платформенно-зависимая библиотека может состоять из следующих компонентов:

1. Потоки;
2. Мьютексы;
3. Семафоры;
4. Очереди сообщений;
5. Таймеры;
6. Менеджеры ресурсов;
7. Объекты динамического выделения памяти;
8. Файлы.

Они реализуются либо на базе программных средств, предоставляемых операционной системой реального времени (ОСРВ), либо непосредственно программистом на базе низкоуровневых запросов к оборудованию. Также возможен вариант, когда платформенно-зависимая библиотека реализуется на базе архитектуры общего назначения в модельном времени.

В случае реализации для ОСРВ данные примитивы используют соответствующие средства API целевой ОСРВ. В модельном времени платформенно-зависимая библиотека включает собственный планировщик (диспетчер) событий, который производит формирование последовательности наступления тех или иных событий, т. е. выполняет функции ядра ОСРВ.

Платформенно-независимая библиотека должна содержать программные примитивы для «изоляции» предметно-ориентированного кода от механизма обмена данными. Она может иметь различную структуру. Целесообразным является архитектура «модулей и каналов», построенная по аналогии со схемотехникой. Программное обеспечение на базе этой архитектуры успешно реализовано в ряде программных продуктов

Модуль – это активный объект структуры, предназначенный для обработки данных. Модуль реализуется на основе потока, включает в себя набор таймеров и очередь сообщений, реализованную на основе клиент-серверной модели.

Канал – это объект структуры, реализующий среду передачи данных. Каналы подключаются к модулям через унифицированные порты. Канал является абстракцией, и может быть реализован на основе любого существующего посредника: сетевых сообщений, внутренних сообщений, буферов в памяти, файлов, пайпов и пр.

Активная работа модуля заключается в обработке сообщений от таймеров и портов, к которым подключен канал. После запуска модуля, он входит в режим ожидания нового сообщения, и по его приходу вызывает соответствующий обработчик, реализующий прикладной уровень создания программы. Сообщения от таймеров и портов поступают в очередь, откуда их читает обработчик общего назначения, который передает обработку одному из двух обработчиков, которые реализуют основную функциональность. Обработчики формируют выходные величины, которые подаются в выходные порты, после чего цикл повторяется.

Подобная реализация платформенно-независимой библиотеки позволяет без изменения кода (а значит внесения изменений в функциональность) произвести моделирование, как в реальном времени, так и в модельном, как на одном компьютере, так и на несколь-

ких, с использованием аппаратных имитаторов, реальной аппаратуры, с вариативной частью (например, пять имитированных колес, и три настоящих).

Рассмотрим применение предложенной архитектуры в рамках системы сопровождения целей, рассмотренной выше. Ясно, что структура программного обеспечения, полученная на этапе системного анализа, естественным образом транслируется в термины данной архитектуры. При этом первый уровень иерархического разбиения дословно соответствует вводимым для его реализации модулям (в терминах данного пункта). Следующий иерархический уровень соответствует реализованным в рамках модулей алгоритмам и осуществляется обычным делением на классы (в случае объектно-ориентированного проектирования) и функции. Применение архитектуры «модули и каналы» для данной системы приведено на рис. 2 (чтобы не загромождать рисунок, входные и выходные порты обозначены вместе как In/Out). Существенным моментом является то, что для параллельных во времени операций предпочтительно использовать отдельный модуль, организуя его взаимодействие с другими при помощи унифицированных портов. Впрочем, возможна ситуация, когда для этого внутри модуля можно ввести отдельные потоки.

Рис. 2. Применение архитектуры «модули и каналы» в рамках системы сопровождения целей

Гибкая организация обменов позволяет заменить каждый модуль его имитационной моделью. В частности, реализовав модуль распределения аппаратных ресурсов и сопряжения с внешними системами в виде моделей реального времени, появляется

возможность осуществлять отладку программного обеспечения на компьютере общего назначения.

Выводы

Значительная функциональная нагрузка и повышенные требования к надежности делают разработку программного обеспечения встраиваемых систем непростой задачей.

На этапе системного проектирования необходимо, прежде всего, правильно декомпозировать программное обеспечение. Структура, сформированная на этом этапе, может быть естественным образом применена в конечном продукте при наличии удобных программных системных примитивов

В силу, зачастую, сложности получения доступа к системе, следует предусмотреть отладочные интерфейсы и отладочную информацию в штатных обменах. Данный вопрос должен решаться на этапе разработки протоколов взаимодействия между модулями программного обеспечения.

Одним из важнейших факторов успешной отладки программ во время разработки и поддержания работоспособности системы при внесении правок является использование программных имитаторов на разном уровне иерархической структуры программного обеспечения. Возможность написания таких имитаторов должна обеспечиваться удобной архитектурой, обеспечивающей обмен данными между модулями программного обеспечения в реальном и модельном времени.

Для полноценной проверки работоспособности встраиваемой системы необходимо использовать стенды полунатурного моделирования, в которые заложена по возможности сложная модель окружающей среды. Это позволяет убедиться в работоспособности программного обеспечения в том числе в нештатных ситуациях, что представляет особый интерес.

Предметно-ориентированный код модулей программного обеспечения встраиваемых систем необходимо делать по возможности переносимым. Это позволяет использовать модульные тесты, которые существенно упрощают отладку и позволяют избежать программных ошибок, а также делает возможным выполнение вычислительной части программ в модельном времени (в частности, для ускорения проверки).

Список литературы

1. Kashif, H. ; Mostafa, M. ; Shokry, H. ; Hammad, S. Model-based embedded software development flow. Design and Test Workshop (IDT), 2009 4th International, DOI: 10.1109/IDT.2009.5404096, 2009.
2. Николаев В.И., Брук В.М. Системотехника: методы и приложения. Л.: Машиностроение, Ленингр. отд-ние, 1985. – 199 с.
3. Эванс Э. Предметно-ориентированное проектирование (DDD): структуризация сложных программных систем. М.: Вильямс. 2011. - 448 с.

4. Embedded Systems. Lee, Edward A. 2002. Vol. 56, London : Academic Press, 2002.
5. Heath, Steve. 2003. Embedded Systems Design. s.l. : Newness, 2003.