RTST - технология программирования встроенных систем реального времени

Парфенов В.В., Терехов А.Н.

1. Введение

1.1. Немного истории

Наш коллектив, костяк которого составляют сотрудники лаборатории системного программирования СПбГУ, в течение многих лет занимался алгоритмическими языками высокого уровня - изучением сравнительных характеристик, реализацией трансляторов, специализированных текстовых редакторов, редакторов связей, отладчиков и т.д. Было реализовано более 20 трансляторов и кросс-трансляторов с языков Pascal, Algol 68, Ada, специализированных языков для многих различных платформ. Поэтому, когда в 1980 году к нам за помощью обратились сотрудники ЛНПО “Красная Заря” (крупнейшее в СССР предприятие по производству телефонных станций различного назначения), работавшие до нас в кодах специализиованных ЭВМ практически без всякой технологической поддержки, первые шаги были очевидны - переход на алгоритмические языки реального времени, использование инструментальных ЭВМ, средств контроля и планирования, подготовка документации на машинных носителях. Действительно общая культура производства заметно улучшилась, но кардинальных сдвигов в производительности труда и качестве получаемого продукта не произошло. Прошло некоторое время пока удалось понять: что традиционных способов программирования для получения успешных результатов не хватает. Постепенно была разработана технология программирования, ориентированная на класс задач в области коммуникации.

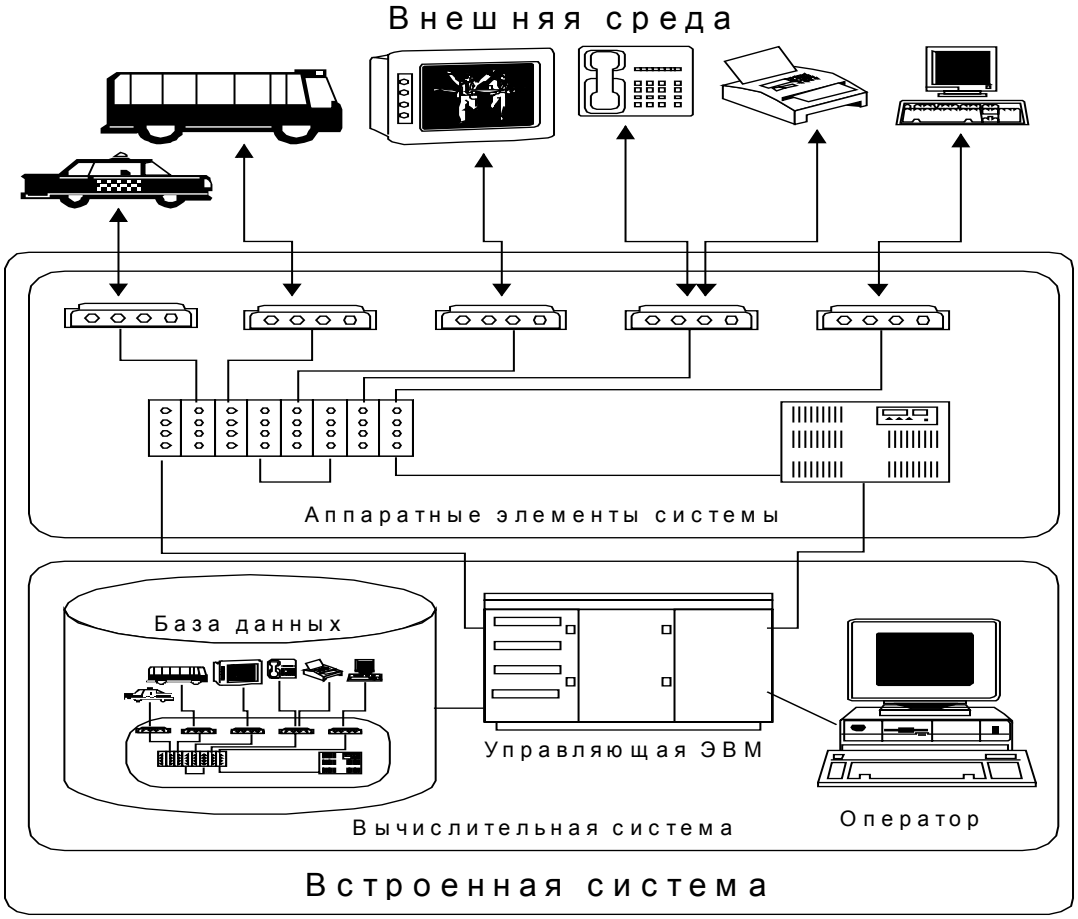
Первые успехи помогли нам получить еще несколько заказов в других областях (сбор и обработка данных в глобальных сетях, управление роботами, самолетами, кораблями, медицинской техникой), что, в свою очередь, позволило оценить применяемые нами средства с различных позиций, понять, какие элементы технологии применимы не только в области телекоммуникации, но и в существенно более широкой области встроенных систем реального времени. Собственно и сама технология, первоначально полностью ориентированная на программное обеспечение телефонных станций, получила название RTST - Real-Time Software Technology.

1.2. Определение встроенной системы

Под встроенной системой в данной работе понимается система, которая

1) погружена во внешнюю среду, частью которой может быть человек (оператор, пользователь);
2) управляется вычислительной системой;

3) представляет собой взаимодействующую совокупность программных и аппаратных элементов;
4) взаимодействует с внешней средой посредством обмена дискретными сигналами, удовлетворяя определенным ограничениям на время приема и обработки входного сигнала и выдачи соответствующих выходных сигналов.



Внешняя среда существует независимо от системы, однако система знает правила ее поведения, умеет адекватно реагировать на информацию, поступающую из внешней среды, и выдавать информацию во внешнюю среду, возможно, оказывающую влияние на ее состояние и поведение. Внешней средой, как правило, является фрагмент реального мира, "обозреваемого" системой. Структурная сложность внешней среды зависит от назначения системы.

Как и внешняя среда, аппаратные средства являются частью реального мира, “обозреваемого” программным обеспечением встроенной системы. От внешней среды их отличает то, что они являются частью самой системы и участвуют в реализации её функций, работая согласованно с программным обеспечением. Одним из основных назначений аппаратных средств является осуществление связи системы с внешней средой. Аппаратные средства обеспечивают прием информации от внешней среды, ее первичную обработку, преобразование в

цифровую форму и передачу программному обеспечению, а также преобразование и передачу информации во внешнюю среду. "Интеллектуальный уровень" аппаратуры может быть различным.

Управляющим элементом встроенной системы является вычислительная среда. Вычислительная среда обеспечивает работу программного обеспечения, хранение информации о состоянии системы и внешней среды, отображение информации на устройствах отображения.

Управление всеми элементами встроенной системы осуществляется программным обеспечением. Программное обеспечение разбивается на *функциональное* и *системное*. Функциональное ПО реализует решение системой возложенных на нее задач. Системное ПО осуществляет управление ресурсами вычислительной среды и обеспечивает к ним доступ со стороны функциональных программ.

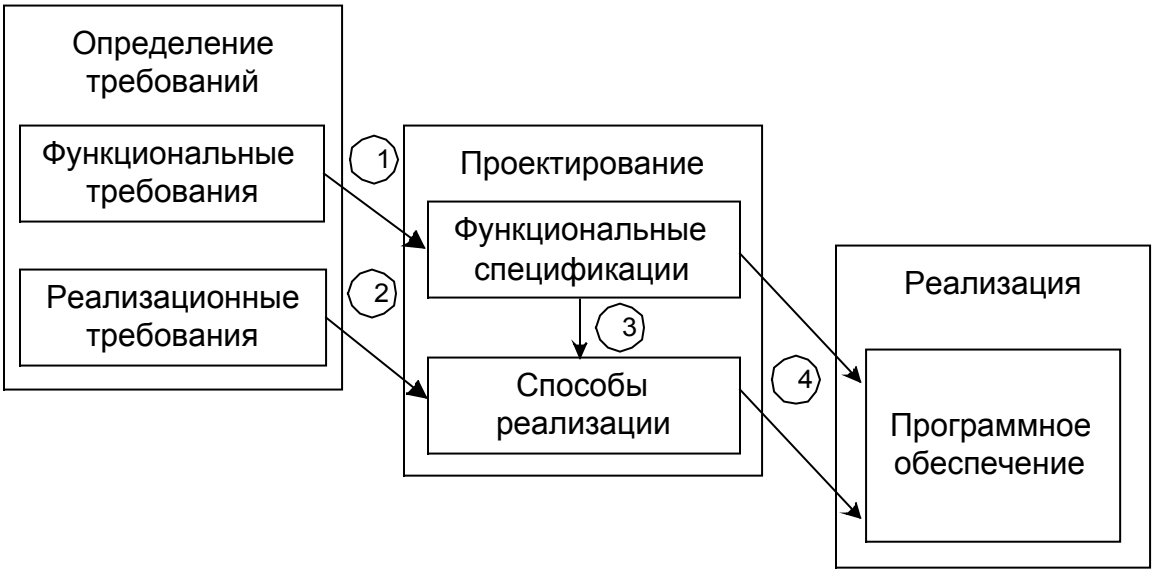
Запуск встроенной системы и ее обслуживание осуществляется человеком (оператором). Есть системы, в которых роль человека практически несущественна (например, человек только включает питание системы), и, напротив, некоторые системы не могут нормально функционировать без активного участия человека (например, системы, функционирующие в среде с постоянно меняющейся структурой, или системы с переменной структурой аппаратных средств).

1.3. Жизненный цикл разработки встроенных систем

Рассмотрим упрощенную схему жизненного цикла встроенной системы [1]. Разработка любой системы начинается с исследования потребностей окружения в этой системе и определения требований к ней. В результате этой деятельности появляется документ, называемый *спецификацией требований*. Спецификация отвечает на вопрос “зачем нужна система?”. Этих требований еще недостаточно для реализации системы. Требования сфокусированы, главным образом, на назначении и роли системы с точки зрения её окружения. Требования часто делятся на две группы: *функциональные* и *нефункциональные* (или *реализационные*). Функциональные требования определяют те задачи, которые должны решаться системой, в то время как нефункциональные требования задают ограничения на реализацию системы (производительность системы, организация её интерфейсов, использование стандартов и пр.). Системы, имеющие различное функциональное назначение, могут иметь одинаковые реализационные требования.

Функциональные требования являются основой для *функционального проектирования,* результатом которого являются *функциональные спецификации*. Функциональные спецификации задают детальные пути выполнения системой требуемых функций без какого-либо учета способов их реализации. На завершающем этапе проектирования на основе нефункциональных требований и функциональных спецификаций происходит отбор *реализационных решений* (или *способов реализации*), которые в совокупности с функциональными спецификациями составляют *технический проект* системы. Реализация программного обеспечения осуществляется с использованием выбранной *системы*

программирования, базирующейся на одном или нескольких *языках программирования* и включающей средства поддержки разработки программ.



Использование на этапе проектирования формальных средств для функциональных спецификаций открывает возможность автоматической реализации программного обеспечения (переход, обозначенный цифрой 4). Поскольку реализация ПО также требует учета нефункциональных требований к системе, то необходимо либо иметь возможность их формальной спецификации, либо попытаться найти набор общих для выбранного класса систем реализационных требований и разработать средства поддержки реализации ПО, настроенные на эти требования.

1.4. Постановка задачи

После приведенных рассуждений сформулируем задачу данной работы следующим образом:

1) Будем рассматривать класс встроенных систем, удовлетворяющих определенным ранее свойствам.
2) Основываясь на свойствах этих систем, установим набор наиболее общих реализационных требований к ним.
3) Исследуем пути удовлетворения этих требований при реализации программного обеспечения системы.
4) Разработаем модель спецификации программного обеспечения встроенных систем и формальные средства, обеспечивающие учет установленных реализационных свойств.
5) Определим структуру средств технологической поддержки этапов жизненного цикла программного обеспечения встроенных систем и содержание каждого из этих этапов.

2. Требования к встроенным системам

Попробуем привести наиболее общие свойства рассматриваемого класса систем, выявленные при решении практических задач, и вытекающие из них требования к встроенным системам.

Сопровождаемость.

Современные системы разрабатываются с учетом их использования в течение достаточно большого срока. В этот период неоднократно возникает необходимость в ее модификации, вызываемая различными причинами (расширение функциональных возможностей, замена аппаратных элементов на более современные, изменение стандартов и правил взаимодействия с внешней средой и т.д.).

Управляемость процессом разработки

Большие встроенные системы, как правило, отличаются значительным объемом и структурной сложностью аппаратных средств и внешней среды и большим набором выполняемых функций. Всё это требует разработки большого объёма программного обеспечения. Достигая определенных границ, сложность программного обеспечения влечет утрату контроля над его разработкой и вынуждает разработчиков совершать ошибки, которые впоследствии достаточно тяжело исправить.

Привлечение к разработке большого числа специалистов влечет за собой достаточно объемную работу по координации их совместной деятельности. Одной из наиболее важных задач здесь является достижение как можно более полного взаимопонимания между различными группами вовлеченных в разработку специалистов. Получаемые на разных этапах описания и спецификации должны обеспечивать их недвусмысленное толкование. Поэтому важная роль отводится используемым формальным средствам спецификации.

Гибкость

Редко бывает так, чтобы при разработке встроенной системы была точно определена структура её аппаратных средств и параметры внешнего окружения. Большинство систем позволяют настраиваться на эти характеристики. Для больших систем такая настройка заключается в создании *базы данных*, содержащей все необходимые параметры. Настройка может выполняться как *статически* до запуска системы, так и *динамически* в процессе её функционирования. Динамическая настройка осложняется тем, что изменения в конфигурации необходимо делать “по живому”, т.е. в данных, которые могут в этот момент использоваться функциональными процессами, что требует согласования между ними и процессом настройки.

Устойчивость

Устойчивость системы означает её готовность реагировать на непредсказуемое поведение внешней среды. Часто от встроенной системы требуется также *отказоустойчивость,* означающая возможность системы продолжать нормальное функционирование в условиях наличия ошибок. К

системам, аварии которых могут нанести вред своему окружению (например, система, управляющая ядерным реактором), могут предъявляться требования *надежности аварийного завершения* (fail safe).

Стабильность

Встроенная система может находиться в одном из двух состояний: *активном* и *пассивном*. В активном состоянии осуществляется функционирование элементов (возможно, не всех) программного обеспечения системы, и система выполняет все или некоторые из своих функций. При этом полное состояние системы описывается состоянием аппаратуры и состоянием данных в вычислительной среде. При переходе системы в пассивное состояние происходит остановка программного обеспечения. При этом не все используемые им данные теряют свою актуальность, часть их должна быть сохранена до следующего запуска системы, что заставляет хранить эти данные в *энергонезависимой* памяти. В качестве средств обработки таких данных выступают системы баз данных, обеспечивающие организацию, хранение и доступ к данным.

Непрерывность

Для большинства систем реального времени является критичным непрерывное и надежное функционирование. Непрерывность функционирования означает невозможность остановки системы, связанную, как правило, с важностью выполняемых ею процессов.

Параллельность

Параллельность является характерным свойством большинства встроенных систем. Наличие аппаратных элементов, работающих в реальном параллельном режиме и одновременное выполнение нескольких функциональных процессов требуют применения параллельных вычислений.

Одна из проблем, связанных с разделением ресурсов в программно-аппаратных системах, заключается в том, что не всегда можно добиться полной их *монополизации*. В частности, это касается данных, отражающих состояние аппаратных элементов системы. Такие данные не могут быть полностью монополизированы функциональным процессом, т.к. существует *аппаратный процесс*, который может потребовать их асинхронного изменения.

Распределенность

При проектировании больших встроенных систем часто используется принцип *модульности.* Это позволяет “собирать” системы различной конфигурации из типовых элементов - *модулей* и, следовательно, увеличивать гибкость системы. Ещё одним следствием модульности является возможность создания *распределенной архитектуры* встроенной системы. При этом каждый отдельный модуль включает в себя свой собственный управляющий элемент - *встроенный процессор*. Все процессоры системы объединяются в *вычислительную сеть*, обеспечивающую обмен информацией между ними.

Понятно, что многие из перечисленных свойств характерны для всех больших программных систем (например, сопровождаемость и управляемость),

другие (например, реальный параллелизм) характерны только для встроенных систем. Важно отметить, что для встроенных систем коренным образом меняется система приоритетов, например, устойчивость и непрерывность нужно обеспечивать даже в ущерб другим характеристикам.

3. Требования к технологии программирования

Для полноты картины приведем требования к технологии разработки программного обеспечения. Здесь вообще все требования применимы для технологии программирования любых больших программных систем, особенностью встроенных систем являются существенно более высокие требования к качеству, производительности и, особенно, надежности применяемых средств.

Наличие целостной методологии

В основе технологии должна лежать система методов и принципов, охватывающая все этапы разработки встроеной системы.

Формальная спецификация проектных решений

Ключевым фактором успешной разработки системы несомненно является ее всеобъемлющая спецификация на всех уровнях абстракции. Использование формальных средств спецификации позволяет повысить эффективность взаимодействия различных групп разработчиков, обеспечивать синтаксический и семантический контроль спецификаций, производить проверку непротиворечивости и полноты спецификаций, использовать автоматизированные средства создания, сопровождения и анализа спецификаций, автоматически поддерживать соответствие спецификаций разных уровней, осуществлять контроль соответствия фазы реализации специфицированным требованиям.

Абстракция

Достаточно большая встроенная система, или даже ее часть, может быть "необозрима" для одного разработчика. В этом случае необходимо использование *абстракций*, позволяющих представить систему в виде набора взаимосвязанных элементов, каждый из которых может быть рассмотрен независимо от остальных.

Прототипирование

Использование формальных спецификаций делает возможным анализ и моделирование альтернативных решений на разных фазах разработки системы. Основой для анализа могут служить "быстрые прототипы", автоматически строящиеся по спецификациям. Технология должна обеспечивать возможность *горизонтального* (когда создается прототип системы на основе только верхних уровней абстракции) и *вертикального* (когда создается прототип для всех уровней абстракции, но для ограниченного числа функций) прототипирования.

Моделирование

По разным причинам часто невозможно выполнить тестирование системы в реальных условиях, особенно на ранних этапах разработки. Следовательно,

технология должна обеспечивать построение *имитационной модели* внешней среды разрабатываемой системы. Такая модель полезна не только для тестирования системы, но и для уточнения и развития требований, изучения свойств системы, понимания логики взаимодействия с внешней средой и т.д.

Автоматизация процесса реализации

Полная формальная спецификация делает возможным автоматизацию процесса реализации системы. Для программного обеспечения это означает генерацию исходных текстов на языке инструментальной и целевой вычислительной системы.

Поддержка сопровождения

Технология должна поддерживать возможность возврата к любой фазе разработки с целью внесения исправлений и дополнений в тексты исходных спецификаций. Технологический процесс не должен содержать шагов, автоматически приводящих к необратимым изменениям в спецификациях.

Тестирование

Тестирование встроенных систем реального времени связано не только с проверкой соответствия разрабатываемой системы установленным функциональным требованиям, но и удовлетворения ею определенных временных ограничений. Это делает процесс отладки достаточно сложным и трудоемким. Важным этапом процесса тестирования является тестирование в инструментальной среде с использованием имитационной модели, работающей в модельном времени.

Возможность настройки

Часто разработка встроенных систем подразумевает разработку не одной системы, а целого класса подобных систем. Системы, принадлежащие одному классу, выполняют одинаковые функции, но различаются количественными параметрами (конфигурация аппаратной части системы и параметры её внешнего окружения). В таких случаях возможно ограничиться разработкой одной *типовой программы*, которая впоследствии может быть настроена на конкретную систему по набору данных, содержащему описание конфигурации аппаратных средств и внешнего окружения.

Возможность повторного использования спецификаций

Отдельные части спецификации готовых систем могут носить вполне самостоятельный характер, т.е. могут быть "отчуждены" от системы, помещены в специальную библиотеку спецификаций и затем использованы при разработке других систем. Это позволяет в значительной степени снизить стоимость будущих разработок, стимулирует принятие хорошо продуманных проектных решений, упрощает вовлечение в работу коллектива новых разработчиков.

Интегрированность

Разработка встроенной системы - сложный и многоэтапный процесс, обеспечиваемый большим набором средств автоматизации, работающий с большим объемом проектной информации, выполняемый большим коллективом

специалистов в области разработки систем и соответствующей прикладной области. Успех в разработке проекта определяется в первую очередь скоординированностью действий всех разработчиков и используемых ими средств. Это требует высокой квалификации руководителей проекта и высокой степени интеграции технологических средств. Основой для интеграции является *единая база данных проекта*, содержащая информацию о состоянии разработки системы.

4. Организация вычислительного процесса

Таким образом, под технологией программирования понимают набор методических, организационных и инструментальных средств, облегчающих создание желаемой программы, помогающих повысить ее потребительские характеристики. Обычно первым шагом проектирования является организация вычислительного процесса, т.е. структура и определение функций всех элементов проектируемого программного комплекса. Считается, что используемая технология программирования должна обеспечить реализацию любой структуры, выдуманной проектировщиком. Однако, все (известные нам) попытки применения столь общего подхода к такой сложной задаче, как ПО встроенных систем реального времени, приводили к неэффективным решениям. По нашему мнению, технология программирования должна отвечать не только на традиционный вопрос “как построить, добиться, оценить”, но и “что мы хотим строить”.

При разработке RTST мы попытались заранее зафиксировать определенные структурные решения, ограничить разнообразие вариантов организации вычислительного процесса, чтобы иметь определенные априорные оценки для сравнения по эффективности возможных вариантов.

Обычно встроенная система реального времени управляется сетью компьютеров, причем управление распределено таким образом, что каждый компьютер делает довольно большую часть работы автономно, т.е. количество действий, выполняемых внутри каждого компьютера существенно больше количества сообщений, пересылаемых по сети. Часто количество элементов сети определяется масштабом конкретной встроенной системы, например, телефонной станции, поэтому желательно организовать вычислительный процесс внутри каждого компьютера также в виде параллельных процессов, взаимодействующих между собой посредством обмена сообщениями, и скрыть от разработчика особенности пересылки сообщений именно по сети, а не между отдельными программами внутри одного компьютера. Отсюда следует естественность организации вычислительного процесса внутри каждого компьютера в виде множества параллельных процессов, взаимодействующих между собой путем обмена сообщениями, а также необходимость отдельной фазы технологии программирования - фазы настройки на конкретный состав оборудования.

Параллельные процессы - классический способ описания поведения сложных систем: независимое управление в каждой подсистеме или элементе оборудования, локализация данных, развитые средства взаимодействия позволяют описывать встроенные системы реального времени наиболее понятным для разработчиков способом. Тем не менее в реальных системах этот способ используется редко из-за низкой эффективности реализации. Дело в том, что в них параллельно существуют тысячи процессов, абсолютное большинство которых

находится в неактивном (“подвешенном”) состоянии, ожидая каких-либо событий или сообщений. Традиционным механизмом реализации параллелизма такого типа является диспетчеризация через короткие промежутки времени по сигналам от таймера (системы разделения времени), однако при этом самой частой операцией становится свертка/развертка процессов - довольно дорогая операция, сильно увеличивающая накладные расходы системы.

Чтобы применить параллельные процессы, но без высоких накладных расходов, мы использовали особенность встроенной системы, состоящую в том, что обычно процессы состоят из очень коротких действий - переходов в терминах расширенной конечно-автоматной модели рекомендаций Z.100 МККТТ [2]. Мы считаем, что если уж процесс получил управление по какому-то входному сигналу, то пусть он доведет переход до конца (до следующего состояния) и только затем передаст управление диспетчеру, который определит, какой из процессов, готовых к исполнению (т.е. получивших сообщения), будет осуществлять переход. Синхронная организация делает возможной простую процедурную реализацию в отличие от традиционной асинхронной, для которой процессы необходимы. Против такой реализации обычно выдвигают два соображения.

Во-первых, часть сообщений приходит по сети от других ЭВМ в непредсказуемые моменты времени. Что ж, в каждой ЭВМ все функциональные процессы соберем в один большой процесс ОС, внутри которого будем использовать синхронную организацию взаимодействия функциональных процессов, а драйверы сети, которые осуществляют буферизацию сообщений (никакой обработки!), будем реализовывать отдельными процессами. Получение сообщения из сети реализуется обработкой прерывания в драйвере, т.е. традиционными дорогими средствами, зато все остальные сообщения внутри одного компьютера (их обычно во много раз больше) реализуются более дешевыми средствами.

Во-вторых, что будет, если один из процессов имеет слишком длинный переход? Тогда можно вставить несколько промежуточных фиктивных состояний (на практике это никогда не встречается).

Такая существенно последовательная и непрерываемая организация переходов позволяет процессам пользоваться глобальными данными (кроме списков входных сообщений, к которым имеют доступ параллельно работающие драйверы сети) без семафоров, критических интервалов и других дорогостоящих средств монополизации ресурсов.

5. Объектно-ориентированный подход к информационному обеспечению встроенных систем

Традиционным подходом к информационному обеспечению встроенных систем реального времени является построение единой БД (централизованной или распределенной), к которой обращаются все процессы через специальные примитивы доступа. В общем случае каждое обращение к некоторой структуре требует ее монополизации на момент обращения, что и неэффективно (несколько обращений к функциям ОС), и ненадежно (одна из самых частых и трудно обнаруживаемых ошибок возникает, когда программист забывает захватить ресурс или, что еще хуже захватывает, но забывает освободить после использования).

Проектирование единой БД обычно ведется только в интересах ПО без учета соответствия данных элементам реального оборудования, что приводит к необходимости довольно сложной генерации данных ПО по исходной спецификации оборудования и сильно затрудняет задачу модификации данных “на ходу”.

Такие особенности встроенной системы, как структурное подобие большей части данных структуре управляемого оборудования и определенная локализация действий вокруг каждого устройства, прямо подталкивают к использованию объектно-ориентированной парадигмы, однако ее эффективное применение вызывает массу практических вопросов.

Во-первых, не вызовет ли разбиение данных на сравнительно мелкие объекты (с точностью до прибора) больших накладных расходов по памяти? Действительно, заголовки процессов в большинстве ОС достигают 100-200 байтов, но, как мы уже отметили, возможны упрощенные синхронные варианты реализации процессов, а в них длину заголовка можно уменьшить на порядок.

Во-вторых, не подменяем ли мы сложные операции доступа к БД не менее дорогими операциями обмена сообщениями? Другими словами, хорошо, если 90% обращений осуществляется к локальным данным объекта и только ради оставшихся 10% приходится посылать сообщения к другим объектам, а если наоборот? Разумеется, на этот вопрос нет универсального ответа, но практический опыт показывает, что обычно удается распределить данные по объектам достаточно удачным образом, т.е. организовать своеобразный конвейер. Сначала управление получает один объект, который, пользуясь своими локальными данными, выполняет определенные действия и посылает сообщение следующему объекту с результатами своей работы в качестве параметра сообщения, т.е. сообщения играют не только информационную, но и управляющую роль. Случаи, когда сообщение посылается только с целью получить значения данных, локализованных в другом объекте, нужно сводить к минимуму. Здесь также можно воспользоваться особенностями встроенных систем, в которых процессы чаще всего далеко не равномерны по приоритетам. Например, в телефонных станциях процессы установления/разъединения соединений критичны по времени, а процессы техобслуживания (которые по объему во много раз больше) - нет. Соответственно, при разбиении системы на объекты нужно учитывать только локализацию данных, нужных для установления и разъединения соединений, а данные, необходимые для техобслуживания, распределять, экономя только собственные усилия.

Несмотря на то, что объектно-ориентированный подход рассматривается в настоящий момент как наиболее обещающая и эффективная парадигма в разработке широкого класса программных систем, опыт ее практического применения в разных областях выявляет наличие большого числа открытых мест, не покрываемых возможностями известных методологий (см. [3], [4], [5]). Этот факт вынуждает разработчиков искать и релизовывать свои средства, расширяющие и дополняющие возможности существующих подходов. В области разработки встроенных систем к числу наиболее существенных недостатков можно отнести остающиеся открытыми вопросы, связанные с объединением объектно-

ориентированной модели с моделью параллельных вычислений, и проблемы спецификации взаимодействий между объектами.

Ограничение видимости интерфейсов

Не все операции, экспортируемые объектом, должны быть видимы всеми его клиентами. Следовательно, желательно иметь возможность спецификации различных интерфейсов класса-сервера для разных классов клиентов, обеспечивающих знание о сервере под разными углами зрения (multiple views). В базах данных для обозначения этого понятия используется термин *подсхема* (subschema), обозначающий проекцию общей схемы данных на нужды конкретного пользователя или класса пользователей.

Экземпляры интерфейсов

Объект-сервер может обеспечивать одновременный доступ к своему сервису со стороны нескольких однотипных клиентов одновременно. В результате, как бы возникает несколько экземпляров одного и того же интерфейса, каждый из которых связан со своим собственным ресурсом.

Интеграция объектно-ориентированных систем с системами управления базами данных

Большинство общеизвестных методов объектно-ориентированного проектирования не учитывают таких аспектов разработки программного обеспечения как *стабильные* данные, *транзакции* и *запросы*. Лишь некоторые из них проводят различие между оперативными, или динамическими, и стабильными, или статическими, данными.

Традиционно, приложения, активно работающие с большими объемами хранимых данных, разрабатываются в виде прикладных программ, использующих средства описания и манипулирования данными, реализованными в некоторой *системе управления базами данных* (СУБД). Такой подход страдает от необходимости одновременного использования двух различных языков с различающимися структурами данных и механизмами управления данными [6].

Координация поведения

Модели взаимодействия параллельных объектов главным образом базируются на механизме обмена сообщениями, который, как средство достаточно универсальное, является, вместе с тем, весьма низкоуровневым, т.к. он может быть использован только для спецификации простых взаимодействий, в которые вовлечено два объекта в один момент времени. Семантика таких взаимодействий не может быть легко расширена. Действительно, посылка сообщения означает только синхронизацию и передачу данных, при этом, например, не фиксируется никаких обязательств сторон.

Определенным развитием в этом направлении можно считать приложение "*теории контрактов*" ([7, 8]), представляющей собой объектно-ориентирование расширение аксиоматики Хоара [9].

Взаимодействие автономных объектов

Взаимодействие между параллельными объектами в большинстве объектно-ориентированных моделей строится на идеологии *клиент/сервер*. Клиент всегда инициирует взаимодействие, первым обращаясь к серверу. При взаимодействии объектов, имеющих в этом отношении равные права, возникают трудности при синхронизации в ситуациях, когда оба равных объекта одновременно начинают взаимодействие. Возникает так называемая "*встречная ситуация*".

Интеграция объектно-ориентированного подхода с моделью параллельных вычислений

Проблеме поиска удовлетворительной модели параллельных объектно-ориентированных вычислений посвящено в последнее время достаточно много внимания ([10, 11, 12). Взаимное притяжение между идеями объектно-ориентированности и параллельности, в первую очередь, обусловлено сходством свойств базовых абстракций, лежащих в их основе: классов и типов процессов.

Одна из сложностей, с которой приходится столкнуться при объединении объектно-ориентированного подхода с параллельными вычислениями - это *аномалия наследования* [13].

Активный объект в параллельной системе характеризуется *состоянием*, от которого зависят алгоритмы большинства его методов. Состояния объектов, представленые значением их локальных переменных, отражают динамическое состояние системы. Смена состояния объекта происходит при обработке им некоторого события, инициированного приемом сообщения.

Рассматривая состояние как существенную часть объектно-ориентированной модели, большинство методов не уделяют достаточного внимания интеграции состояния с наследованием. Остается непонятным, должна ли спецификация состояния наследоваться подклассом. Если нет, то значит состояние каждого подкласса должно определяться заново, что означает отсутствие переиспользования - основного выигрыша от наследования. Если состояние наследуется и, следовательно, может быть специализировано, то необходимо представлять, как это скажется на методах надкласса, которые используют общую структуру состояния. В общем случае, это должно повлечь переопределение этих методов, что также означает отказ от переиспользования кода.

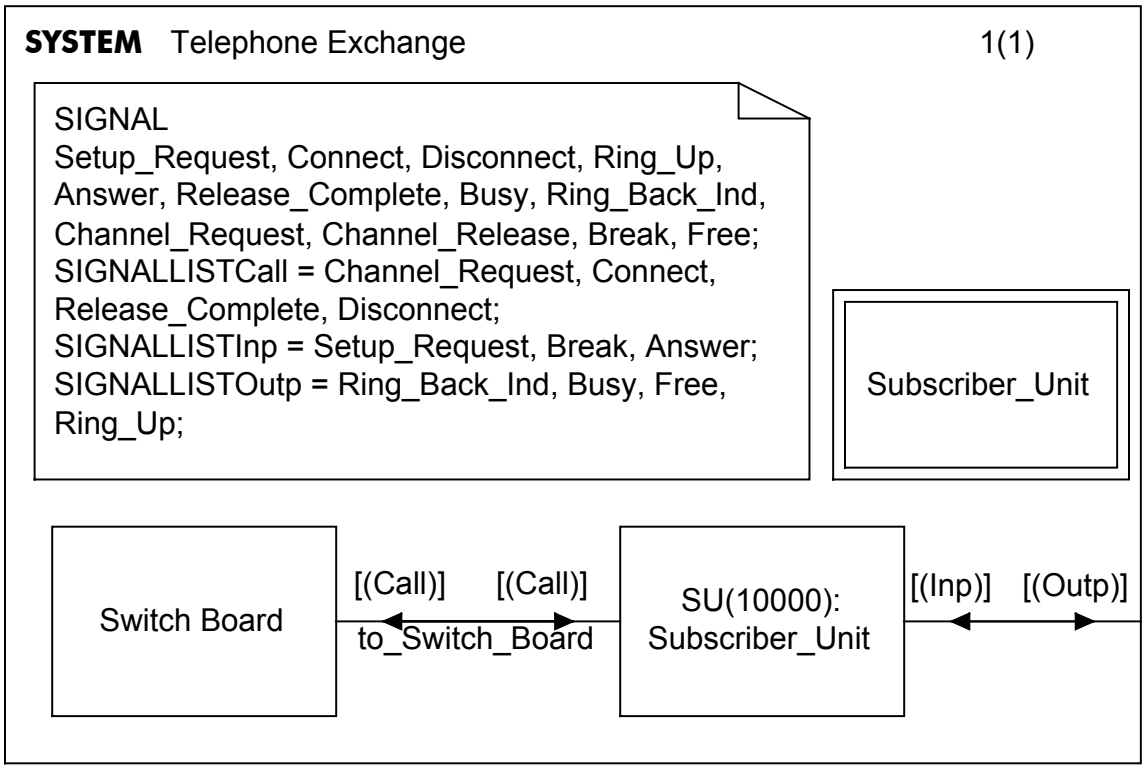
6. Графические средства проектирования встроенных систем

Большинство известных методологий проектирования программного обеспечения рассматривают понимание пользователем на ранних этапах свойств разрабатываемой системы как ключевой фактор успеха при их разработке, значительно сокращающий затраты на ее доработку по результатам испытаний, опытной эксплуатации и внедрения.

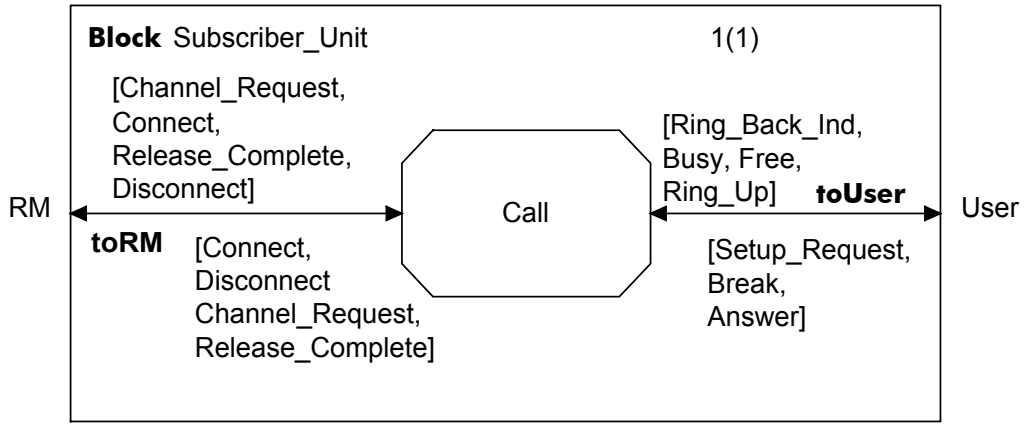
Программная инженерия является достаточно сложным видом деятельности, и в настоящее время потрачено большое количество усилий на разработку подходов, снижающих ее сложность. Одним из таких шагов можно считать введение визуальных средств проектирования и спецификации, которые обладают большей выразительной мощностью по сравнению с традиционными

текстовыми языковыми средствами [14]. "Визуальные языки используют невербальные пути восприятия разработчика, комбинируя средства интерактивной графики с формальными языками. Общепризнанно, что механизмы человеческой памяти, в большей степени, ориентированы на восприятие зрительной информации, и его уровень значительно выше при исследовании графических связей в сложных картинках, чем при чтении линейного текста" [15]. Графические нотации, широко применяемые в различных инженерных дисциплинах, эффективно используются и для описания архитектурных аспектов разрабатываемых программных систем и схем их щповедения.

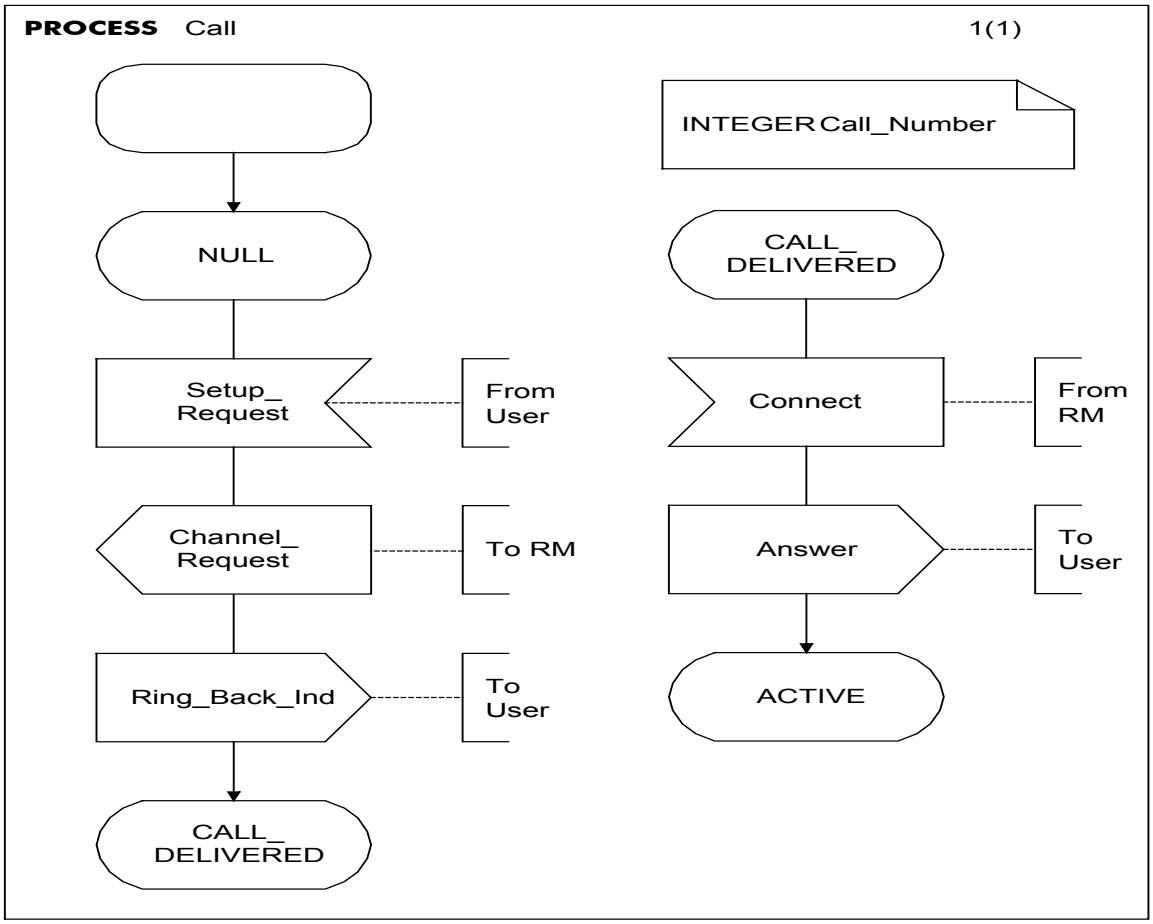
Графические средства для исчерпывающего задания поведения системы предложены в языке описаний и спецификаций SDL [2], представляющем собой законченную методологию для спецификации встроенных систем. В соответствии с SDL, встроенная *система* представляется в виде набора *блоков*, связанных между собой *каналами*, обеспечивающими информационный обмен блоков между собой и с внешним окружением системы.



Каждый блок, в свою очередь, может быть разбит на *подблоки*, т.е. рассмотрен как подсистема.



На самом нижнем уровне декомпозиции блок разбивается на *процессы*, связываемые с входящими в блок каналами и между собой *маршрутами сигналов.* Процессы являются активными элементами системы и представляются последовательностью действий над своими локальными данными и данными включающих их блоков. Процессы используют маршруты и каналы для обмена информацией между собой. Зафиксировано два способа взаимодействия между процессами: посылка асинхронного сообщения (*сигнала*) и вызов удаленной процедуры. Кроме взаимодействия через маршруты, возможно обращение к статически видимому процессу, т.е. локализованному в одном из включающих данный процесс блоков, или обращение по динамическому идентификатору процесса, переданному параметром в сигнале.



Процесс в SDL - это расширенный конечный автомат. Состояние автомата описывается набором точек ожидания (стационарных состояний) внешних стимулов процесса, которые представлены сигналами, получаемыми от других процессов, сигналами от часов системы об истечении установленных временных интервалов (timeout) и вызовами удаленных процедур процесса. Кроме этого, состояние процесса уточняется произвольной, сколь угодной сложной, структурой данных. Переход в расширенном конечном автомате осуществляется из одного стационарного состояния в другое при получении внешнего стимула. При этом алгоритм перехода может быть достаточно сложным, т.к. включает анализ не только стационарного состояния, но и внутренних данных процесса.

Используемая для задания переходов графическая нотация представляет собой модификацию *блок-схем*, дополненную символами состояния, приема и посылки сигнала, вызова удаленной процедуры, порождения процесса и пр.

Особое место в языке SDL отведено спецификации данных. Помимо предопределённых типов данных, традиционных *конструкторов* данных, таких как создание структурного значения, и предопределенных *генераторов*, возможно задание определённых пользователем генераторов и атомарных абстрактных типов данных.

Определение абстрактного типа задается множеством литеральных значений (констант), набором операций над значениями и аксиом, заданных в

терминах литералов и операций. Аксиомы неявно задают семантику операций в декларативной форме. Все предопределенные типы данных также определены в описании SDL, как абстрактные.

Нотация, используемая в SDL, допускает как формальную спецификацию, так и полуформальное описание процесса. Во втором случае выдерживаются только синтаксические правила, задающие последовательность графических символов, внутри которых может быть записан любой неформальный текст. Формальная же спецификация позволяет полностью описать все управляющие процессы. По такой спецификации может быть сгенерирован текст на выбранном языке реализации, как это, например, делается в технологии SDT (SDL Design Tool), использующей в качестве языков реализации CHILL и C++.

К числу недостатков SDL-методологии можно отнести ряд деталей, усложняющих полную автоматизацию процесса реализации:

1) необходимость достаточно жесткого задания структуры системы в спецификации;
2) неэффективность реализации спецификаций абстрактных типов данных;
3) неоднозначность декомпозиции сигналов;
4) недостаточность спецификации различных типов связей между процессами и блоками.

7. Предлагаемая модель описания и реализации программного обеспечения встроенных систем

В поисках путей преодоления приведенных выше недостатков объектно-ориентированной методологии и SDL-методологии с целью повышения эффективности как технологического процесса разработки, так и поучаемого ПО мы, с одной стороны, ввели некоторые ограничения, приблизившись к так называемой объектно-базированной модели, с другой стороны, ввели некоторые дополнительные возможности. К сожалению, ограничение на объем данной работы не позволяют нам привести всех обоснований нашего выбора. В данном разделе мы остановимся лишь на основных свойствах базовых понятий используемой модели.

Базовыми понятиями модели являются *система, объект, линия подключения, виртуальный канал, сообщение* и *элемент данных*. Объект служит для представления любой потенциально активной сущности системы, линия подключения соответствует статической связи между двумя активными сущностями, виртуальный канал отражает динамическую связь между объектами, сообщение является элементарной единицей информационного обмена между ними, элемент данных используется для представления пассивных сущностей.

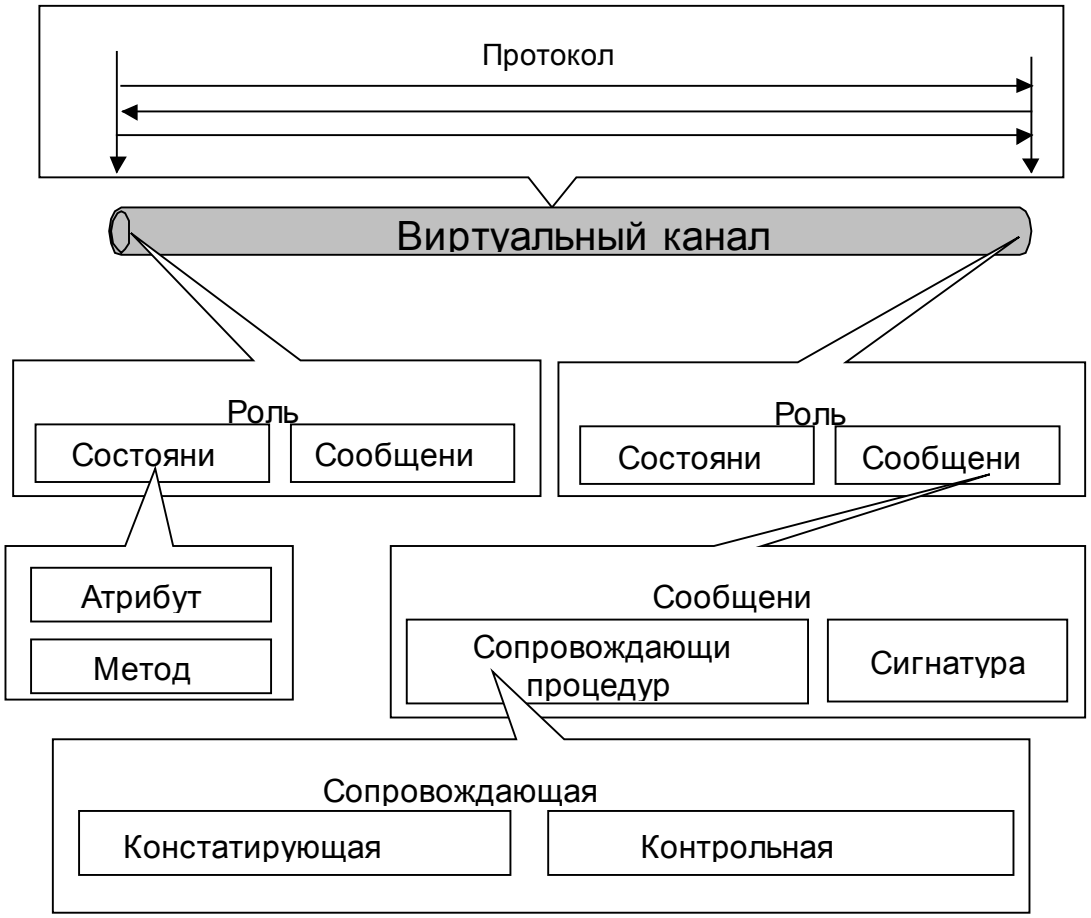
Ко всем базовым понятиям применимы понятия *класса* (или *типа*) и экземпляра. Класс задает общие характеристики для всех своих экземпляров. Далее при обсуждении свойств каждого понятия не всегда уточняется о чем идет речь, о классе или об экземпляре, предполагая, что это должно быть очевидно из контекста.

Класс системы задается спецификацией всех типов понятий, используемых при построении ее экземпляров. Центральным свойством системы, характерным для рассматриваемого класса применений, является статичность ее архитектуры. Система представляется в виде набора объектов, жестко связанных между собой

линиями подключения. Любой объект системы находится либо в активном, либо в пассивном состоянии. *Активный* объект представляет собой процесс, обрабатывающий локальные данные объекта и взаимодействующий с процессами других объектов через связывающие их линии подключения посредством механизма обмена сообщениями. Линии подключения, с которыми связан объект, составляют все его знание о структуре и свойствах своего окружения. Ни один объект системы не может управлять ее структурой, т.е. создавать и удалять другие объекты, устанавливать и разрушать связи между ними. Состояние системы разбивается на *статическую* и *динамическую* составляющие. Статическую картину составляют данные, существующие независимо от того, активна или нет система. Управление статической картиной осуществляется извне некоторой *административной системой*. Динамическая картина отражает состояние исполняемых объектами процессов и их взаимодействий. В ее изменение вносит вклад любой активный объект системы.

Элементарной единицей взаимодействия между двумя объектами является *сообщение*. Класс сообщения задается сигнатурой, определяющей типы передаваемых параметров и, возможно, тип возвращаемого результата. Наличие результата и его тип определяют модель взаимодействия, реализуемую посылкой сообщения: *асинхронная*, *синхронная* или *удаленный вызов процедуры*.

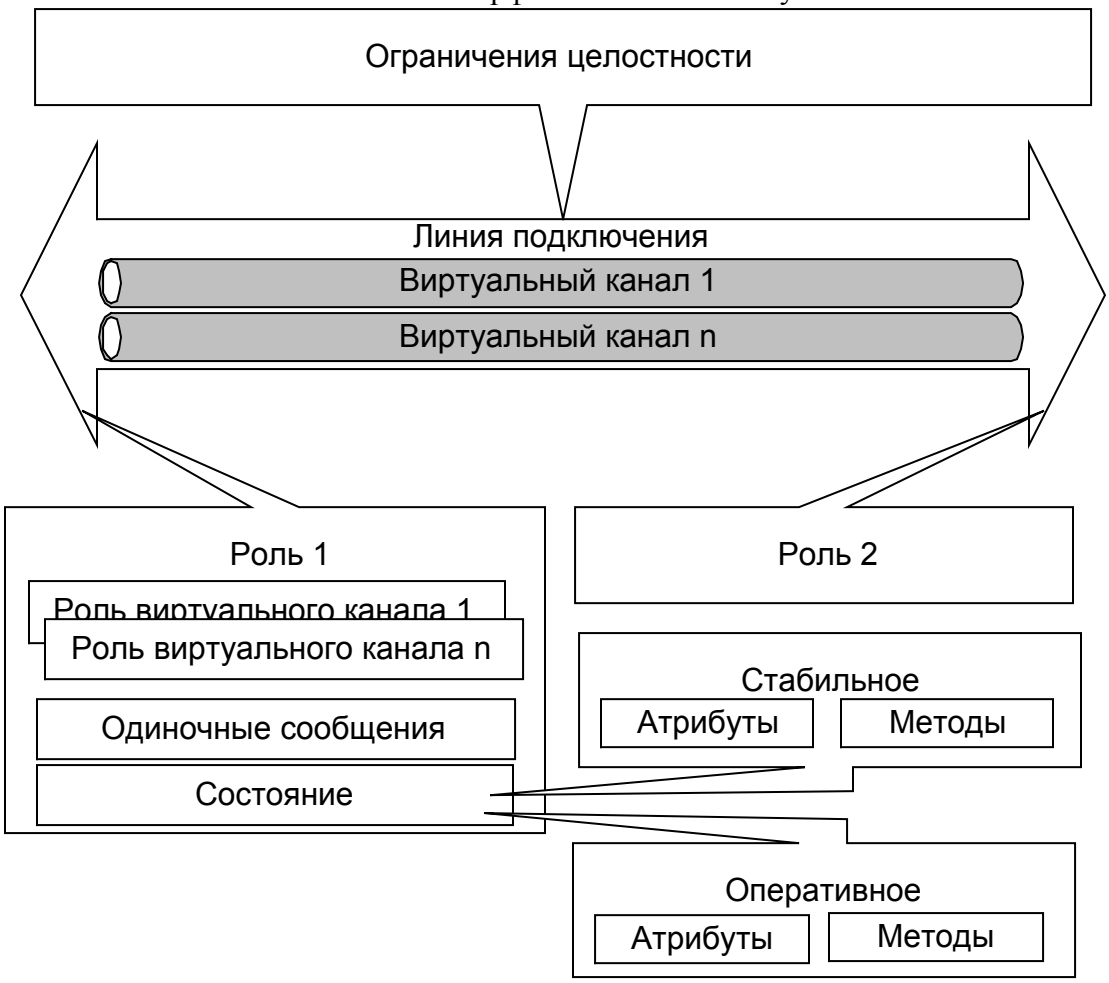
Виртуальный канал позволяет установить долговременную динамическую связь между двумя объектами, используя которую они могут осуществить последовательный обмен группой логически связанных сообщений. Виртуальный канал имеют два конца, каждому из которых может быть приписана своя *роль*. С ролью связывается набор сообщений, которые могут быть приняты и обработаны на соответствующем конце виртуального канала. В зависимости от того, совпадают или различаются роли концов виртуального канала он может быть, соответственно, однополюсным или двухполюсным. С каждым концом виртуального канала может быть связана некоторая структура данных, отражающая динамику взаимодействия. Элементами этой структуры являются атрибуты и методы, выполняющие вычисления на этих атрибутах.



С каждым сообщением могут быть связаны *сопровождающие процедуры*: *процедура посылки*, вызываемая на конце-отправителе в момент посылки сообщения, и *процедура приема*, вызываемая на конце-получателе в момент приема сообщения. Каждой из этих процедур доступны параметры сообщения и данные соответствующего конца виртуального канала. Сопровождающая процедура разбита на две части: *констатирующую* и *контрольную*. В констатирующей части могут быть выполнены любые вычисления над данными конца виртуального канала. Контрольные части процедур посылки и приема сообщения задают условия, которые должны быть удовлетворены на соответствующих концах виртуального канала в момент их вызова. Динамика взаимодействия по виртуальному каналу задается *протоколом*, описывающим возможные последовательности обменов сообщениями между его ролями. Взаимодействие по виртуальному каналу напоминает диалог между двумя объектами, который начинается открытием канала и завершается его закрытием. Открытие и закрытие осуществляются неявно посылкой специально выделенных для этих целей сообщений. Роли равноправны, т.е. не разделены явно на ведущую (*клиент*) и ведомую (*сервер*), и виртуальный канал может быть открыт любой из них. При открытии виртуального канала создаются связанные с его концами структуры данных, которые инициализируются констатирующими частями сопровождающих процедур сообщения, открывшего канал. По открытому виртуальному каналу

объекты могут обменяться любым числом сообщений в порядке, регламентируемом протоколом.

Обмен по виртуальному каналу представляет собой более сложную форму взаимодействия объектов, чем простой обмен одиночными сообщениями. Спецификация класса виртуального канала задает одновременно информационное наполнение диалога, условия и правила его ведения, средства отображения и обработки его состояния. Класс виртуального канала соответствует некоторому существующему в системе логическому протоколу, который может быть использован для спецификации интерфейсов большого числа объектов. Каждый из объектов может внести свои уточнения в общую спецификацию, но внешние свойства этого интерфейса останутся неизменными.

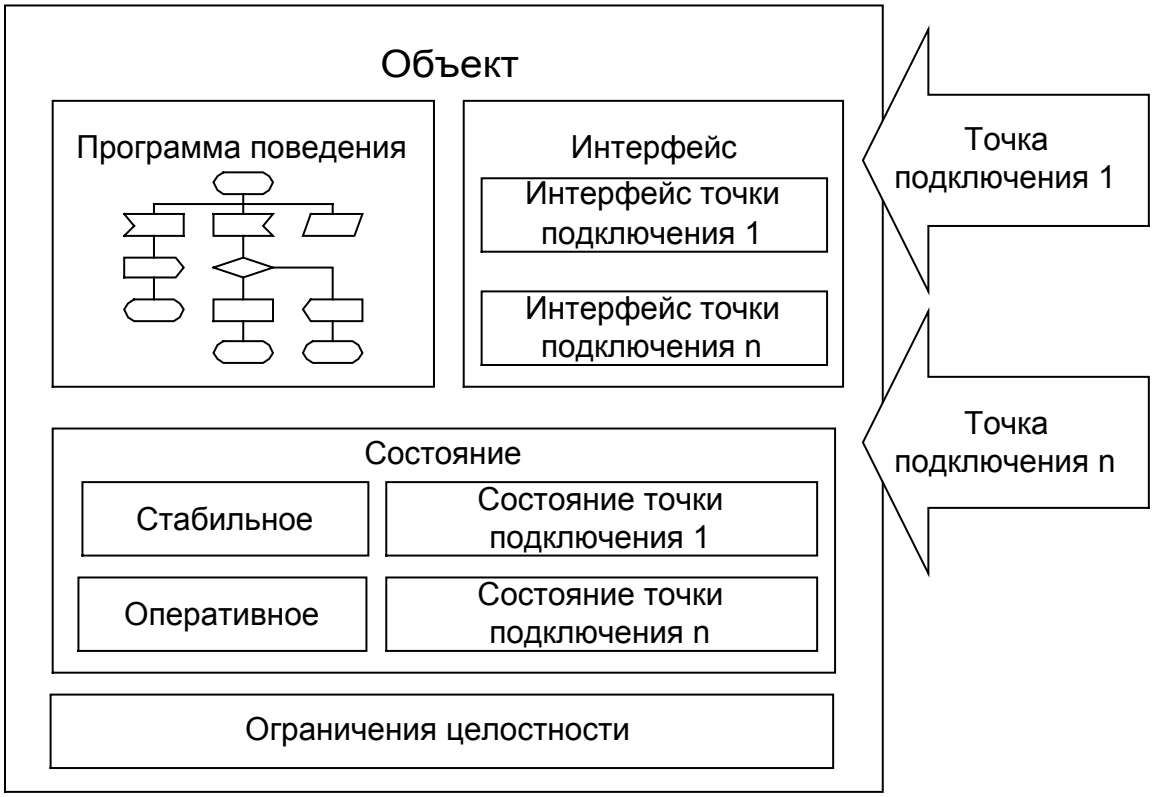


Классы *линий подключения* определяют типы внешних интерфейсов между активными сущностями системы и служат для отражения статических связей между объектами. Линия имеет два конца, каждому из которых, как и в случае виртуального канала, может быть приписана своя *роль*. Содержание каждой роли определяется внутренней структурой линии подключения, которая задается набором *виртуальных каналов* и *одиночных сообщений*.

Одиночные сообщения служат для обмена информационными и управляющими воздействиями, не накладывающими дополнительных обязательств

(например, необходимость ответной реакции) на взаимодействующие стороны. Виртуальные каналы линии подключения задают множество диалогов, которые могут быть установлены между объектами, соединенными этой линией. Все диалоги одной линии подключения независимы друг от друга и могут быть открыты одновременно. Каждый виртуальный канал линии подключения представляет собой специализацию некоторого класса виртуального канала, заданного в спецификации системы. Они могут дополнять структуры состояний концов канала, переопределять их методы и сопровождающие процедуры сообщений.

С каждой ролью линии подключения также связывается структура данных, атрибуты и методы которой разбиты на две группы: *стабильные* и *оперативные*. Стабильные данные отражают статические характеристики линии подключения, сохраняемые между активными состояниями системы. Динамические данные отражают динамику взаимодействия между связанными линией объектами. В терминах стабильных данных задаются *ограничения целостности*, которые должны быть удовлетворены любым экземпляром линии подключения.



Класс *объекта* характеризуется структурой данных, отражающей его *состояние*, программой, задающей его *поведение*, и точками подключения, определяющими его *интерфейсы*. Точки подключения используются для соединения объектов между собой. Каждая точка характеризуется классом включаемой в нее линии и одной из ролей, что определяет множество принимаемых и передаваемых через эту точку сообщений. Программа поведения описывается *расширенным конечным автоматом*, состоящим из фиксированного

набора *состояний* и набора *переходов*, задающих реакции объекта на сообщения, приходящие в точки подключения. Реакции заключаются в обработке внутренних данных объекта и передаче ответных сообщений. Программы поведения активных объектов функционируют параллельно, синхронизируясь через обмен сообщениями по связывающим их точкам подключения.

Данные объекта состоят из его собственных данных, которые разделены на *стабильные* и *оперативные*, и данных его точек подключения, делегируемых концами включенных в них линий. *Ограничения целостности*, записанные в терминах стабильных данных, задают условия, которые должны быть удовлетворены любым объектом класса.

В заключение заострим внимание на одной из отличительных особенностей предлагаемой модели. Как в большинстве объектно-ориентированных моделей, так и в SDL-модели, при определении связи между объектами (блоками в SDL) отношение (канал в SDL) определяется как способ связи между двумя конкретными типами объектов (блоков). В RTST же при спецификации линии подключения не задаются типы связываемых ею объектов. Вместо этого при определении типов объектов задаются типы линий подключения, используемых для связывания экземпляров определяемого типа с другими объектами. Этим удается промоделировать наследование объектами общих интерфейсов без введения полного механизма наследования, включающего и наследование поведения, неприменимого в нашей модели из-за аномалии наследования.

8. Обеспечение отказоустойчивости и учет занятых ресурсов

Среди наиболее критичных требований к встроенным системам выделяются *отказоустойчивость* и *непрерывность* их функционирования. Как было отмечено ранее одним из характерных свойств встроенных систем является одновременное существование в системе нескольких параллельных процессов. Это требует решения задачи разделения ресурсов.

При процесс-ориентированном подходе каждый существующий в системе процесс соответствует решению некоторой функциональной задачи. Он порождается управляющей программой при запуске задачи и уничтожается при её окончании. Для выполнения требуемых функций процесс захватывает необходимые ему ресурсы из общего пула глобальных ресурсов, используя один из известных механизмов монополизации ресурсов. Завершение процесса или его уничтожение влечет освобождение всех захваченных им ресурсов. Аварийное завершение процесса также требует освобождения всех захваченных ресурсов. Освобождение ресурса во встроенной системе включает в себя: освобождение представляющей его структуры данных, приведение значения данных в целостное состояние и приведение в некоторое исходное состояние соответствующего ресурсу аппаратного элемента (если последний существует). При нормальном завершении процесса освобождение ресурсов, как правило, не вызывает проблем. При аварийном завершении освобождение структуры данных может быть возложено на соответствующий системный механизм монополизации ресурсов, приведение данных в исходное состояние выполняется некоторой реакцией на исключительную ситуацию, а приведение аппаратного элемента в исходное

состояние, по идее, должно породить специальный параллельный процесс (т.к. это действие, включающее взаимодействие с реальным физическим объектом, может оказаться достаточно сложным). Недостатки этого подхода очевидны. Во-первых, текст функционального процесса перегружается массой технических деталей, связанных с установкой процедур обработки исключительных ситуаций. Во-вторых данные о состоянии используемого ресурса могут быть распределены по данным функционального процесса, например скопированы в стек процесса, состояние которого в результате сбоя может быть некорректным. В-третьих, захватывая ресурс, процесс полностью берет на себя обработку всех связанных с ним сигналов, поступающих из внешней среды, некоторые из которых могут не иметь отношения к функциональному процессу. В-четвертых, возможность использования ресурса в разных функциональных процессах влечет необходимость повторения соотвествующего программмного кода в каждом из них. В-пятых, такой подход может стимулировать появление разных способов реализации прямых (безотказных) и обратных (возникающих при отказах) веток функциональных алгоритмов. Последнее значительно осложняет реализацию программного обеспечения, перенося центр тяжести алгоритмизации обработки сбоев с этапа проектирования ПО на этап его реализации.

В нашей модели нет понятия функционального процесса и, следовательно, нет явного динамического распределения ресурсов. Все ресурсы системы статически распределены по объектам. Аналогично и вся функционалистика системы распределена по объектам. При участии объекта в некотором функциональном процессе он сам “занимает” имеющиеся у него данные, изменяя их состояние. В процессе участия значения данных могут быть неоднократно изменены, отражая историю этого участия. При завершении функционального процесса данные переводятся в состояние, которое условно может быть названо исходным. При этом происходит неявное их освобождение. Свойство инкапсуляции означает, что только объект знает структуру и правила работы со своими данными и, следовательно, только он может восстанавливать их корректное состояние. Начало функционального процесса, как правило, инициируется получением некоторого внешнего стимула из физической среды, и, в первую очередь, обрабатывается объектом, соответствующим породившему этот стимул элементу. Поочередно передавая управление связанным объектам, функциональный процесс протекает по ним, занимая необходимые ресурсы. Здесь, как и при процесс-ориентированном подходе, нормальное завершение функционального процесса не вызывает особых проблем. Обмениваясь друг с другом сообщениями, объекты оповещают друг друга о вхождении функционального процесса в заключительную стадию и параллельно с этим освобождают занятые под него ресурсы. Аварийное завершение функционального процесса может быть вызвано двумя причинами: сбой в одном из используемых аппаратных элементов или ошибка в программе одного из объектов. Надо отметить, что во встроенных системах, как правило, аппаратный сбой не является неожиданной (ошибочной для ПО) ситуацией, если имеются аппаратные средства поддержки обнаружения сбоя.

При обнаружении программного сбоя целесообразно прекратить выполнение программы поведения аварийного объекта, а остальные объекты

оповестить о необходимости аварийного завершения функционального процесса и освобождении занятых под него ресурсов. Для этого необходимо знать обо всех динамических взаимодействиях объекта. В качестве средства учета этих взаимодействий используются виртуальные каналы.

Существование открытого виртуального канала означает наличие у связанных им объектов общего (возможно, виртуального) ресурса, что, как правило, выражено тем, что некоторые данные этих объектов находятся в промежуточном, нецелостном, состоянии. Знание этого факта позволяет зафиксировать зависимости между данными взаимодействующих объектов и использовать эту информацию для согласования статической и динамической картин при управлении конфигурацией функционирующей системы и минимизации последствий программных сбоев.

При запуске нового функционального процесса участвующие в нем объекты попарно соединяются виртуальными каналами, фиксирующими образующиеся динамические связи.

Появление ошибки в программе одного из объектов приводит к переводу этого объекта в пассивное состояние. Такое действие не имеет последствий для других объектов системы, если этот объект не установил с ними динамических связей, т.е. не открыл виртуальных каналов. В противном случае, связанные объекты могут “зависнуть” в состоянии постоянного ожидания сообщения от остановленного объекта и задерживать свои ресурсы и ресурсы, связанных с ними объектов, которые могли бы быть использованы в других функциональных процессах. В результате, может быть заблокировано много полезных ресурсов системы и снижена ее пропускная способность. Чтобы этого не произошло, системная процедура перевода объекта в пассивное состояние включает рассылку специально выделенных аварийных сообщений во все открытые объектом виртуальные каналы. Необходимо отметить, что во встроенных системах ожидание такого рода сообщения является вполне естественным, т.к. вероятность отказа входящей в состав системы аппаратуры достаточно высока. Обработка отказа при аппаратном сбое отличается лишь тем, что рассылку сообщений, закрывающих взаимодействие, организует объект, отражающий в системе состояние вышедшего из строя аппаратного элемента.

9. Описание технологического процесса

На качество встроенной системы в значительной степени влияет качество процесса ее разработки. Возможность аттестации (validation) спецификаций системы на ранних этапах заметно уменьшает число итераций при доработке по результатам испытаний и снижает стоимость последующих модификаций системы. Возможность генерации управляющей программы по спецификации системы позволяет провести отладку спецификаций на имитационных моделях в инструментальной среде. Для построения имитационных моделей окружения системы ее спецификация дополняется спецификацией составляющих окружение классов объектов и типов их интерфейсов. Моделирование внешней среды может быть достаточно детальным.

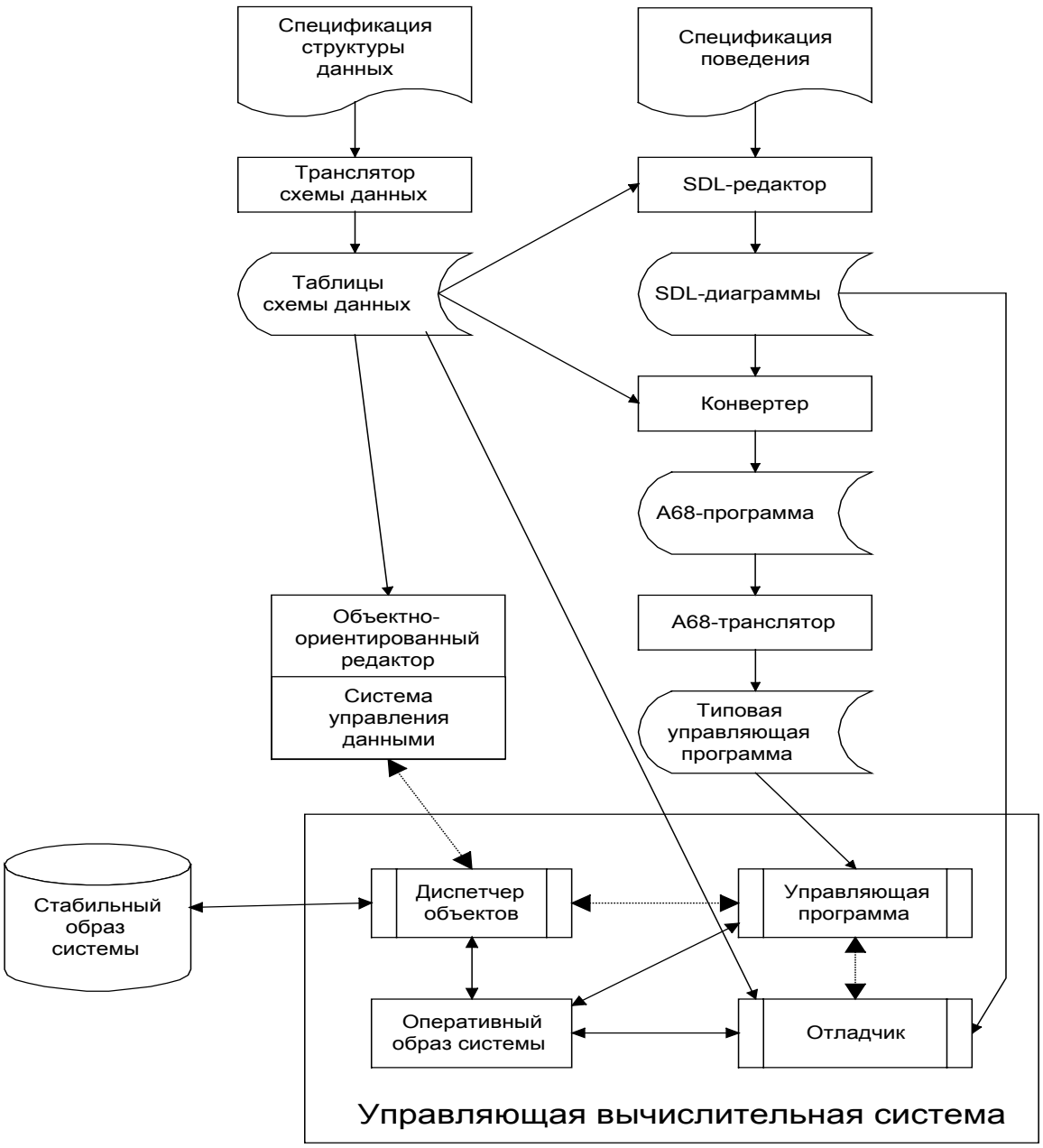
Основным источником динамической противоречивости спецификаций является нарушение динамики взаимодействий. Проконтролировать ее

корректность позволяет учет заданных в спецификации условий, содержащихся в контрольных частях сопровождающих процедур сообщений и в протоколах виртуальных каналов.

Спецификация класса системы используется для построения типовой программы, управляющей любым экземпляром класса. Настройка программы на конкретный экземпляр заключается в создании стабильного образа системы, существующего независимо от ее активности и сохраняемого во внешней памяти. Процесс настройки заключается в создании объектов, редактировании их стабильных данных и соединении их линиями подключения. При редактировании данных объекта выполняется изменение значений их атрибутов и вызов их методов. Редактирование объекта считается завершенным, если выполняются заданные для его класса ограничения целостности. При установлении соединения точек подключения двух объектов линией проверяется их совместимость по типам и ролям, а также удовлетворение заданных ограничений целостности. Настройка системы может проводиться как до активизации системы, так и в момент ее функционирования. В режиме функционирования выполняется разрешение конфликтов между *внутренними*, выполняемыми программами объектов, и *внешними*, управляющими конфигурацией, *транзакциями*. Если объект участвует в некотором функциональном процессе, о чем говорит наличие у него открытых виртуальных каналов, то внешняя транзакция приостанавливается до его освобождения или объект принудительно переводится в пассивное состояние, имитируя возникновение программного сбоя. После завершения редактирования, объект снова может быть активизирован.

Проектирование, настройка и эксплуатация программного обеспечения встроенной системы в соответствии с приведенным подходом осуществляется набором *интегрированных технологических средств* [16, 17].

Этап проектирования поддержан средствами, обеспечивающими формальную спецификацию алгоритмического и информационного обеспечения встроенной системы в объектно-ориентированном стиле и построение на ее основе рабочей программы и управляющих таблиц базы данных, работающими на инструментальной ЭВМ.



Спецификация структуры объектно-ориентированной базы данных осуществляется на специализированном *языке описания схемы данных* (ЯОСД). Текст описания на ЯОСД представляет собой набор описаний типов данных, типов объектов, типов линий подключения и типов виртуальных каналов.

Обработка текста описания на ЯОСД осуществляется *транслятором схемы данных*, который выполняет синтаксический и семантический анализ описаний и строит управляющие таблицы, содержащие схему данных.

Спецификация алгоритмов поведения объектов осуществляется на специализированном языке SDLA68, основанном на графической нотации SDL, с зафиксированным синтаксисом текстов, записываемых в графические символы.

Синтаксис текстов основывается на понятиях, заложенных в базовый язык (АЛГОЛ 68) и в язык описания схемы данных.

Программное обеспечение встроенной системы представляет собой набор SDL-диаграмм, описывающих программы поведения объектов. SDL-диаграмме доступны локализованные внутри соответствующего объекта данные (статические и динамические). Набор принимаемых и передаваемых сообщений определяется множеством точек подключения объекта и множеством виртуальных каналов, доступных через эти точки подключения.

Ввод SDL-диаграмм, описывающих поведения объектов, осуществляется специализированным *редактором SDL-диаграмм*. SDL-редактор осуществляет проверку корректности построения SDL-диаграммы (правильная последовательность и сочетание символов) и формирует файл с ее внутренним представлением.

После подготовки текстов программ SDL-редактором они обрабатываются *конвертором*, который, используя таблицы, содержащие спецификацию объектно-ориентированной схемы данных, выполняет синтаксический анализ текстов, записанных в SDL-символы, и преобразует тексты диаграмм в набор текстов на АЛГОЛе 68, составляющих рабочую программу.

Для каждого типа объекта порождается набор процедур, реализующих генерацию экземпляра объекта в оперативной памяти управляющей ЭВМ и программу поведения объекта.

Тексты рабочей программы транслируются с АЛГОЛа 68 в коды целевой управляющей ЭВМ и комплексируются вместе со средствами динамической поддержки в загрузочный модуль.

Этап настройки и эксплуатации поддержан средствами, включающими *диспетчер объектов*, *систему управления базой данных* и *объектно-ориентированный редактор базы данных*, функционирующими на целевой ЭВМ.

Рабочая программа, сгенерированная по формальным спецификациям схем объектов и их программ поведения, является типовой управляющей программой для целого класса встроенных систем, построенных в соответствии с объектно-ориентированной схемой системы. Настройка типовой программы на конкретное приложение осуществляется посредством формирования статической картины базы данных, содержащей описание конфигурации аппаратных средств встроенной системы и ее внешней среды.

Управление объектно-ориентированной базой данных осуществляется системой управления базой данных (СУБД). В ведении СУБД находятся задачи создания, уничтожения, соединения и разъединения объектов, коррекции статических параметров объектов.

Поддержка функционирования программ поведения объектов осуществляется диспетчером объектов. Диспетчер объектов осуществляет последовательный запуск программ поведения готовых к исполнению объектов и управляет обменом сообщениями между связанными объектами. По запросам от СУБД диспетчер объектов осуществляет генерацию и уничтожение экземпляров объектов, запуск и остановку их программ поведения, соединение и разъединение объектов.

Объектно-ориентированный редактор осуществляет управление процессом интерактивного создания и редактирования статической картины объектно-ориентированной базы данных. Объектно-ориентированный редактор предоставляет возможность в процессе настройки и эксплуатации системы осуществлять занесение информации в базу данных, внесение коррекций и поиск информации в базе данных. В процессе редактирования данных редактор осуществляет исчерпывающий контроль корректности и непротиворечивости данных, допуская построение базы данных лишь в точном соответствии со схемой данных, построенной по спецификации схемы объектов.

10. Заключение

Представленная в статье технология разработки программного обеспечения встроенных систем RTST разработана и развивается объединенным коллективом сотрудников СПбГУ, ГП “Терком”, ИСИ СО РАН, НИИ “Дельта” НПК “Красная Заря”. Авторы статьи выражают благодарность всем, кто принимал участие в выработке и обсуждении изложенных решений, реализации технологических программных средств и отладке технологической среды. Особого упоминания заслуживает вклад внесенный в данную работу П.С. Лавровым, М.А. и А.А. Бульонковыми, С.Л. Алексеевой, А.В. Ведерниковым, Н.Н. Вояковской, Т.С. Мурашовой, А.Н. Ивановым, А.А. Цепляевым, Б.А. Федотовым, Ю.К. Лавровой.

11. Библиография

1. *Braek, R., Haugen, O.* **Engeneering Real-Time Systems**. Prentice Hall International (UK) Ltd, 1993.

2**. CCITT Recommendation Z.100: CCITT Specification and Description Language (SDL)** // COM X-R 26, ITU General Secretariat, Geneva, 1992.

3. *Aksit, M., Bergmans, L.* **Obstacles in Object-Oriented Software Development** // Proceedings of OOPSLA-92, Oct 1992, pp 341-358.

4. *Fichman, R., Kemerer, C.* **Object-Oriented and Conventional Analysis and Design Methodologies** // IEEE Computer, Vol 25, No 10, Oct 1992, pp 22-39.

5. *Wirfs-Brock, R., Johnson, R.* **Surveying Current Research in Object-Oriented Design** // Communications of ACM, Vol 33, No 9, Sept 1990, pp 104-124.

6. *Bloom, T., Zdonik, S.B.* **Issues in the Design of Object-Oriented Database Programming Languages** // Proceedings of OOPSLA-87, Oct. 1987, pp 116-132.

7. *Meyer, B.* **Applying Design By Contract** // IEEE Computer, Vol 37, No 10, Oct 1992, pp 40-51.

8. *Meyer, B.* **Sequential and Concurrent Object-Oriented Programming** // Proceedings of TOOLS'2, Anghor/SOL, Paris, June 1990, pp 17-28.

9. *Hoare, C.* **An Axiomatic Basis of Computer Programming** // Communications of ACM, Vol 12, No 10, Oct 1969, pp 576-583.

10. *Meyer, B.* **Systematic Concurrent Object-Oriented Programming** // Communications of ACM, Vol 36, No 9, Sept 1993, pp 56-80.

11. *Agha, G.* **Concurrent object-oriented programming** // Communications of ACM, vol 33, no 9, Sept 1990, pp 125-141.

12. *Lieberman, H.* **Concurrent Object-Oriented Programming in Act 1** // Object-Oriented Concurrent Programming, MIT Press, Cambridge, Mass., 1987.

13. *Matsuoka, S., Wakita, K., Yonezawa*, *A.* **Inheritance in Concurrent Object-Oriented Language** // Proceedings of 7th Japan Society for Software Science and Technology, 1990, pp 65-68.

14. *Shu N.C.* **Visual Programming**, Van Nostrand Reinhold Company, New York, 1988.

15. *Reader G.* **A Survey of Current Graphical Programming Techniques,** IEEE Computer, 1985, pp.11-25.

16. *Ëàâðîâ Ï.Ñ., Ïàðôåíîâ Â.Â., Òåðåõîâ À.Í.* **Îáúåêòíî-îðèåíòèðîâàííûé ïîäõîä â ÀÌÒÑ ñ ïðîãðàììíûì óïðàâëåíèåì** // Êà÷åñòâî ïðîãðàììíîãî îáåñïå÷åíèÿ: ìàòåðèàëû IV Ìåæäóíàðîäíîé êîíôåðåíöèè. - Äàãîìûñ. - 1992.

17. *Ïàðôåíîâ Â.Â.* **Îáúåêòíî-îðèåíòèðîâàííûé ïîäõîä â ïðîåêòèðîâàíèè ïðîãðàììíîãî îáåñïå÷åíèÿ âñòðîåííûõ ñèñòåì** // Ïðîáëåìû òåîðåòè÷åñêîãî è ýêñïåðèìåíòàëüíîãî ïðîãðàììèðîâàíèÿ. - Íîâîñèáèðñê. - 1992 - Ñ.158-173.