RTST - технология программирования встроенных систем реального времени

Терехов А.Н,
Санкт-Петербургский Государственный Университет,
ГП Терком

Аннотация

В данной работе описывается технология RTST, с помощью которой было успешно реализовано несколько телекоммуникационных систем. На основе анализа специфики предметной области - систем реального времени – были предложены несколько ограниченные, но существенно более эффективные по сравнению с традиционными подходами организация вычислительного процесса и информационного обеспечения целевой системы, средства обработки сбоев и откатов, стратегия учета и освобождения ресурсов.

1 Введение

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 специализированных ЭВМ практически безо всякой технологической поддержки, первые шаги были очевидны - переход на алгоритмические языки высокого уровня, использование автоматизированных 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. И постепенно была разработана технология программирования, ориентированная на задачи в области телекоммуникации.

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.

RTST родилась в процессе работы по реализации ПО телефонной станции правительственной связи. Эти работы завершились успешно, технология доказала свою полезность и эффективность, однако этот эксперимент был не вполне чистым, поскольку технологию использовали ее авторы. Тем не менее она приобрела некоторую известность в России. Специалисты Центрального НИИ связи (г.Москва), столкнувшись с трудностями сопровождения и недостаточной производительностью используемых ЭВМ, решили заменить ЭВМ и одновременно перейти на нашу технологию в своей полностью цифровой телефонной станции С-32, к тому времени уже сертифицированной и серийно выпускавшейся. Менее чем за полгода основной объем работ был выполнен, причем авторы технологии только давали консультации, но ничего не разрабатывали сами.

Еще более сложную проверку технология выдержала при разработке ПО телефонной станции “Бета”, выполненной нашим коллективом в течение 1996 года. Опять-таки станция уже была сертифицирована и было выпущено более 20 штук на двух заводах в России и Белоруссии, но оказалось, что она очень сложна в установке и сопровождении. Столь большой объем выполненных работ не позволил даже ставить вопрос о замене вычислительных средств, хотя их

низкая производительность вызывала большие сомнения в возможности использования технологии. Однако и эта работа была выполнена успешно. Поскольку большинство технических трудностей взяла на себя технология, разработчики управляющего ПО получили возможность сосредоточиться на решении действительно принципиальных вопросов организации вычислительного процесса. Полученное в результате ПО оказалось существенно эффективнее исходного, сделанного вручную. Главными результатами явились резкое снижение затрат на сопровождение и автоматическая генерация данных для каждой конкретной станции.

2 Организация вычислительного процесса

Технологией программирования мы называем набор методических, организационных и инструментальных средств, облегчающих создание программы и помогающих повысить ее потребительские характеристики. 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 заранее зафиксировали определенные структурные решения и тем самым ограничили разнообразие вариантов организации вычислительного процесса.

Параллельные процессы - классический способ описания поведения сложных систем: независимое управление в каждой подсистеме, локализация данных, развитые средства взаимодействия позволяют описывать системы 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. 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, сильно увеличивающая накладные расходы системы.

Чтобы эффективно использовать параллельные процессы, мы использовали особенность встроенных систем, состоящую в том, что обычно ее процессы имеют очень короткие действия - переходы в терминах расширенной конечно-автоматной модели рекомендаций Z.100 МККТТ [1]. Мы считаем, что если процесс получил управление по какому-то входному сигналу, то он должен довести переход до конца (до следующего состояния) и только затем передать управление диспетчеру, который определит, какой из процессов, готовых к исполнению (т.е. получивших сообщения), будет работать следующим. Синхронная организация параллелизма делает возможной простую процедурную реализацию в отличие от традиционного асинхронного подхода, при реализации которого необходимы процессы. Против синхронной реализации обычно выдвигают два соображения.

Во-первых, часть сообщений приходит по сети от других ЭВМ или оборудования в непредсказуемые моменты времени. Для решения этой проблемы соберем все функциональные процессы 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 все-таки имеет слишком длинный переход? Тогда можно вставить несколько промежуточных фиктивных состояний (на практике это встречается крайне редко).

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.

3 Объектно-ориентированный подход к информационному обеспечению встроенных систем

Традиционным подходом к разработке ПО для встроенных систем является построение единой базы данных (централизованной или распределенной), к которой все процессы обращаются через специальные примитивы доступа. В общем случае 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 базы данных обычно ведется 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 во время функционирования системы (“на ходу”).

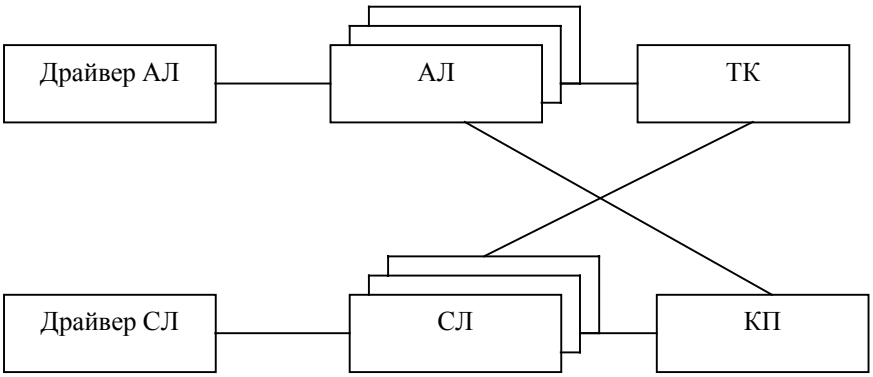
Такие особенности встроенных систем, как структурное подобие большей части данных структуре управляемого оборудования и определенная локализация действий вокруг каждого устройства прямо подталкивают к использованию объектно-ориентированной парадигмы, однако ее эффективное применение вызывает массу вопросов.

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, поэтому представление объектов в виде полноправных процессов ОС действительно очень накладно. Но при определенных ограничениях возможны упрощенные синхронные варианты реализации объектов, при этом длина заголовка уменьшается на порядок.

Во-вторых, не подменяем ли мы сложные операции доступа к БД не менее дорогими операциями обмена сообщениями? Другими словами, хорошо, если 90% процентов запросов осуществляется к локальным данным объекта и только ради 10% приходится посылать сообщения, а если наоборот? Разумеется, на этот вопрос нет универсального ответа, но практический опыт показывет, 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 получения данных, локализованных в другом объекте, нужно сводить к минимуму. Здесь также можно воспользоваться особенностями встроенных систем, в которых процессы чаще всего далеко не равноправны по приоритетам. Например, в телефонных станциях процессы установления/освобождения соединения 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.

В качестве примера рассмотрим упрощенную телефонную станцию.



АЛ – Абонентская Линия;
СЛ – Соединительная Линия;
ТК – Телефонная Книга;
КП – Коммутационное Поле.

Рис. 1. Упрощенная телефонная станция

Здесь каждому абоненту соответствует объект типа АЛ, каждой линии,соединяющей данную станцию с другой, - объект типа СЛ. Объект типа ТК является справочником номеров абонентов данной телефонной станции и хранит данные, необходимые для маршрутизации. Объект типа КП соответствует реальному коммутационному полю. Драйверы играют связующую роль с внешним миром и обычно реализуются в виде ассемблерных программ, вызываемых по прерываниям.

Когда абонент поднимает трубку, соответствующий объект АЛ получает сообщение, принимает все цифры номера и посылает сообщение объекту ТК с принятым номером в качестве параметра. Заметим, что все это время АЛ работает только со своими локальными данными. ТК осуществляет поиск вызываемого абонента также только по своим локальным данным и посылает сообщение найденному объекту типа АЛ или СЛ. Вызываемый абонент или соединительная линия по своим локальным данным определяет, занят он или свободен, и если свободен, то посылает собщение КП на проключение. КП по своим локальным данным находит свободную линию и т.д.

4 Обеспечение отказоустойчивости и учет занятых ресурсов

Среди наиболее критичных требований к встроенным системам выделяются отказоустойчивость и непрерывность их функционирования, а также тщательный учет используемых ресурсов.

При процесс-ориентированном подходе каждый существующий в системе процесс соответствует решению некоторой функциональной задачи. Для выполнения требуемых функций он захватывает необходимые ему ресурсы из общего пула глобальных ресурсов, используя один из известных механизмов монополизации. 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 в целостное состояние части глобальных данных системы, использующихся процессом, и перевод в некоторое исходное состояние соответствующего ресурсу аппаратного элемента (если таковой 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. Во-вторых, захватывая ресурс, аварийный процесс полностью берет на себя обработку всех сигналов, поступающих из внешней среды. В-третьих, возможность использования ресурса в разных функциональных процессах влечет необходимость повторения соотвествующего программмного кода в каждом из них.

В нашей модели нет понятия функционального процесса и, следовательно, нет явного динамического распределения ресурсов. Все ресурсы системы статически распределены по объектам, что достаточно естественно: трудно себе представить, что в управляемой системе физические приборы будут часто появляться и исчезать - обычно все приборы устанавливаются и связываются кабелями при запуске системы, изредко эта конфигурация может меняться, но относительно часто какие-нибудь приборы могут быть заблокированны, например, для ремонта. Аналогично, и вся функционалистика системы распределена по объектам. При участии в некотором функциональном процессе сам объект свободно изменяет имеющиеся у него данные, меняя свое состояние. После завершения функционального процесса данные переводятся в состояние, которое условно может быть названо исходным. 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 им ресурсы. Аварийное завершение функционального процесса может быть вызвано двумя причинами: сбоем в одном из используемых аппаратных элементов или ошибкой 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 соответствующего функционального процесса и освобождения занимаемых им ресурсов. Для этого необходимо знать обо всех динамических взаимодействиях объекта. В качестве средства учета этих взаимодействий используются виртуальные каналы.

Виртуальный канал – это средство динамической связи двух объектов. При запуске нового функционального процесса участвующие в нем объекты попарно соединяются виртуальными каналами, фиксирующими образующиеся динамические связи. 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.

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. Чтобы этого не произошло, системная процедура перевода объекта в пассивное состояние включает рассылку специально выделенных аварийных сообщений во все открытые объектом виртуальные каналы. Обработка отказа при аппаратном сбое отличается лишь тем, что

рассылку сообщений, закрывающих взаимодействие, организует объект, отражающий в системе состояние вышедшего из строя аппаратного элемента.

5 Описание технологического процесса

Итак, создаваемую систему удалось разбить на объекты и зафиксировать организацию вычислительно процесса. На самом деле, удачное разбиение - весьма нетривиальный творческий процесс, что заставило нас уже после нескольких выполненных проектов приступить к развитию технологии “вверх” с упором на начальные этапы разработки (о дальнейшем развитии RTST см. [2]).

Разработка системы с помощью технологии RTST начинается с описания на специальном языке схемы объектов (в языках программирования принято говорить “тип”, а не “схема”). Сначала объект описывается как черный ящик (точки его подключения к другим объектам, перчень входящих и исходящих сообщений и их параметры). Затем описываются внутренние атрибуты объекта. Они могут быть двух видов - те, которые не могут быть изменены самим объектом (статические), и обычные рабочие переменные (динамические).

С помощью транслятора схем такие спецификации объектов преобразуются во внутреннюю схему данных, которая является “сердцем” системы.

Поведение объекта специфицируется с помощью расширенного конечного автомата языка SDL/GR [1]. SDL является обобщением языка блок-схем с возможностью специфицировать структурную декомпозицию разрабатываемой системы, описывать параллельные процессы и их поведение.

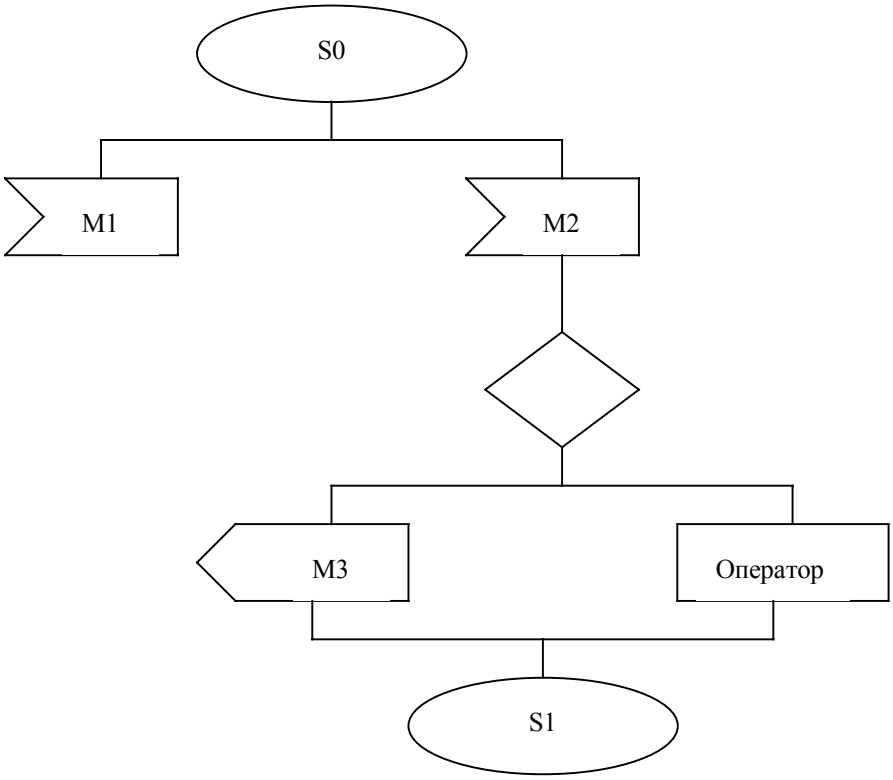


Рис. 2. Пример SDL-диаграммы.

В примере, изображенном на рис. 2, объект в состоянии S0 ожидает сообщение M1 или M2. Получив сообщение M2 объект совершает переход: выполняет проверку некоторого условия и, в зависимости от результата, делает определенное действие (Оператор) или посылает сообщение M3. После этого объект переходит в состояние S1. В RTST существует графический редактор для спецификации поведения объекта в таком виде, который, кроме того, осуществляет проверку синтаксической корректности спецификации и формирует ее

внутреннее представление. (В дальнейшем конечно-автоматные диаграммы SDL будем называть просто SDL-диаграммами).

С помощью конвертора SDL-диаграммы превращаются в текст на алгоритмическом языке (ранее Алгол 68, теперь С), при этом происходят дополнительные проверки на соответствие со схемой объектов, по которой также происходит генерация служебных процедур (посылка сообщения, генерация экземпляра объекта в оперативной памяти и т.д.).

Тексты, полученные в результате конвертации, транслируются в коды целевой управляющей ЭВМ (или, в целях отладки, в коды инструментальной ЭВМ) и собираются вместе со средствами динамической поддержки в загрузочный модуль.

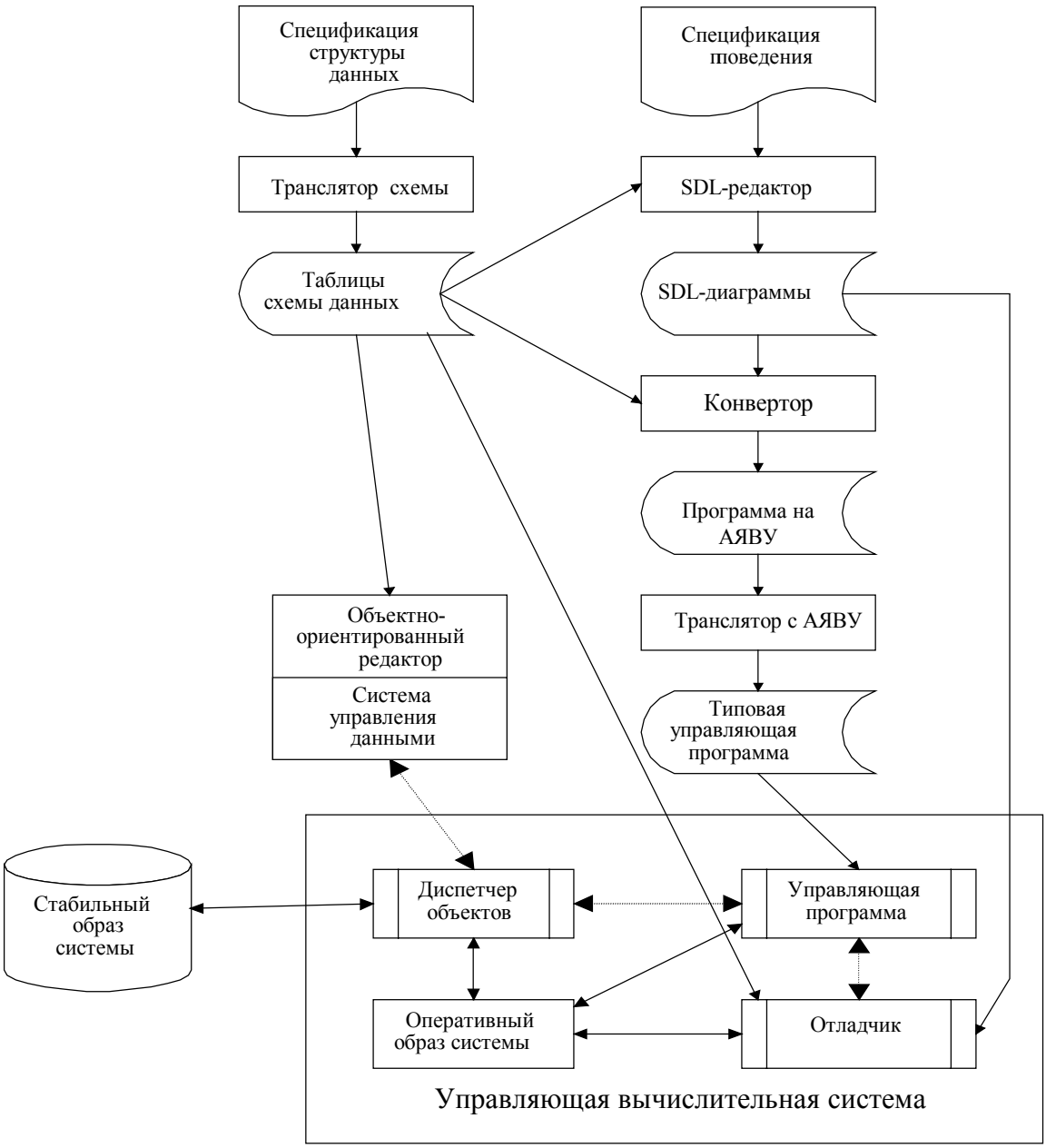
Настройка типовой программы на конкретное приложение осуществляется посредством формирования статической базы данных, содержащей описание конфигурации аппаратных средств встроенной системы и ее внешней среды.

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. Для повышения реактивности системы в случае рестартов СУБД реализована по методу “тщательного замещения”, когда внутри транзакции ни один блок не пишется на старое место.

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

Объектно-ориентированный редактор осуществляет управление процессом интерактивного создания и редактирования статической базы данных объектов. Он предоставляет возможность при настройке и эксплуатации системы осуществлять занесение информации в базу данных, внесение изменений и поиск. 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. Все формы ввода генерируются автоматически по базе данных схем.

В нашей технологии не различаются этапы первоначального ввода данных и их исправления в процессе функционирования системы. С помощью объектно-ориентированного редактора можно, не мешая работе системы, создать или уничтожить какие-то экземпляры объектов, изменить значения их атрибутов, изменить какие-то связи. Только если потребовалось переопределить схему хотя бы одного объекта, придется перезапустить всю систему, но на приктике это встречается очень редко.



6 Заключение

Представленная в статье технология RTST разработана и развивается объединенным коллективом сотрудников СПбГУ, ГП “Терком”, ИСИ СО РАН, НПО “Красная Заря”. Автор статьи выражает благодарность всем, кто принимал участие в выработке и обсуждении изложенных решений, реализации технологических программных средств и отладке технологической среды. Особого упоминания заслуживают В.В. Парфенов, П.С. Лавров, М.А. и А.А. Бульонковы, С.Л. Алексеева, Н.Н. Вояковская, А.Н. Иванов, Д.В. Кознов, А.В. Лебедев, Т.С. Мурашова, Н. Ф. Фоминых, А.А. Цепляев.

7 Литература

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

[2] А. Иванов, Дм.Кознов, А.Лебедев, Т. Мурашова, В.Парфенов, А.Терехов. Объектно-ориентированное расширение технологии RTST. См. настоящий сборник.

[3] Лавров П.С., Парфенов В.В., Терехов А.Н. Объектно-ориентированный подход в АТМС с программным управлением // Качество программного обеспечения: материалы IV Международной конференции. Драгомыс. 1992.

[4] Парфенов В.В. Объектно-ориентированный подход в проектировании программного обеспечения встроенных систем реального времени. // Проблемы теоретического и экспериментального программирования. Новосибирск. 1992.