сопровождается физическим и виртуальными видеофрагментами с конкретизацией моментов, отображающих суть работы;

- расширенная тестовая система самопроверки – проверка усвоения учебного материала в виде вопросов и зарезервированных ответов, предоставляемая слушателю специальной интерактивной системой. Тестовая система разбита на четыре укрупненных модуля в соответствии с требованиями Болонской системы обучения в высшей школе.

Применение разработанных программных продуктов, необходимых при самостоятельном изучении дисциплины (при выполнении виртуальных лабораторных работ, при текущем тестировании, решении задач, самопроверки знаний), а также применения дешевых в производстве электронных носителей информации вместо дорогих бумажных, позволит получить значительный экономический эффект от внедрения результатов разработки в учебном процессе высших учебных заведений.

Таким образом, использование мультимедийных технологий в учебном процессе вуза по направлению «Электроника» позволяет перейти от пассивного к активному способу реализации образовательной деятельности, при котором обучающийся является главным участником процесса обучения.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Смолянинова О.Г.* Мультимедиа для ученика и учителя // ИНФО. – 2002. – № 2. – С. 48–54.
2. *Смолянинова О.Г.* Мультимедиа в образовании (теоретические основы и методика использования): Монография. – Красноярск: Изд-во КрасГУ, 2002. – 300 с.
3. *Петрунин Ю.Ю.* Искусственный интеллект как феномен современной культуры // Вестник Московского университета. – М.: 1994. №8. – С. 28–34.

Ю.А. Веретельников

ОБ ОДНОМ ПОДХОДЕ К ПОСТРОЕНИЮ УПРАВЛЯЮЩЕГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ ВСТРАИВАЕМЫХ СИСТЕМ РЕАЛЬНОГО ВРЕМЕНИ

Каждый год миллионам микропроцессоров и микроконтроллеров находится применение в качестве центрального управляющего модуля как для персональных компьютеров, так и для так называемых «интеллектуальных» устройств. В их число теперь входят и бытовые (например, принтер или стиральная машина, современная технологичная игрушка, мобильный телефон), и промышленные устройства (системы сбора и контроля параметров промышленных объектов, системы диагностики). Эти системы принято называть встраиваемыми или встроенными. Под встроенной компьютерной системой понимается совокупность аппаратного и программного обеспечения, являющаяся неотъемлемой частью более крупной системы и используемая для управления и/или непосредственного мониторинга

этой системы с применением специальных аппаратных устройств[1]. Обратим внимание, насколько гибким является такое определение – как электронная «начинка» домашнего электроприбора, так и сложный субблок в системе управления атомной электростанцией одинаково хорошо подпадают под него.

Их можно охарактеризовать как системы с жёсткими ограничениями на массу, габаритные и энергетические показатели, зачастую устанавливаемые в непосредственной близости от контролируемого объекта. Технический прогресс приводит к тому, что цифровое управление и цифровые системы охватывают всё большие области человеческой деятельности. Неудивительно, что число таких встраиваемых систем на сегодняшний день превосходит число персональных компьютеров в сотни и тысячи раз. Об этом знают специалисты, занимающиеся разработкой и внедрением подобных устройств.

В настоящее время традиционным стал подход, который обусловливает наличие цифрового блока в составе системы; более того, им, как правило, является универсальный, программно управляемый цифровой блок – микропроцессор или (что чаще) микроконтроллер. По известным причинам такая схема является наиболее предпочтительной [2].

Несмотря на ограниченные по сравнению с компьютерами общего назначения возможности, к встраиваемым системам предъявляются жёсткие требования относительно потребления энергии, времени реакции на внешние события, предоставляемые возможности. Не случайно появился класс систем ***реального времени***, обеспечивающий особые показатели по вышеперечисленным требованиям.

Для обеспечения функционирования сложных систем цифровой обработки сигналов (ЦОС), таких, например, как системы, реализующие протоколы беспроводной передачи данных в режиме реального времени, требуются специальные, адаптированные под определенные приложения, аппаратные конфигурации. Состав и вид конкретной конфигурации, как правило, зависит от специфики решаемой задачи, однако каждая из них включает цифровой блок.

Независимо от состава конфигурации, встраиваемые системы (кроме, может быть, самых простых) из-за наличия управляемого процессорного элемента требуют для своего успешного функционирования специального программного обеспечения – управляющей среды.

Тем не менее, от каждого устройства требуется своя, часто уникальная, функциональность. Вместе с многообразием аппаратных конфигураций это ещё одна причина, по которой управляющее программное обеспечение в каждом случае создается с учетом целевой платформы.

Классическим подходом к построению такого управляющего программного обеспечения был и по-прежнему остается подход, использующий в той или иной степени модель ***«жесткого цикла»***. Так называется модель вычисления качественных и количественных характеристик отклика системы на заданное воздействие, в которой основу алгоритма работы устройства составляет цикл; на каждом шаге его опрашиваются все источники входных данных для получения исходных величин, результатом об-

работки которых является выходная величина. Система предстает перед нами в виде двух моделируемых сущностей – объекта управления и регулятора. (Напомним, что речь здесь идет о цифровом, т.е. дискретном регуляторе.) В качестве вспомогательного, но не отнюдь не обязательного средства используется математический аппарат теории цифровых конечных автоматов (ЦКА).

Классический подход, совместно с привлечением методики ЦКА, нашел успешное применение для решения большинства разнообразных задач, возникающих перед встраиваемыми системами.

В числе его достоинств – полная временная детерминированность при корректно построенной модели, возможность строго математического описания ее поведения. Подход, привлекающий ЦКА, дает возможность описать поведение системы привычными терминами реального мира.

Тем не менее при всех его теоретических положительных качествах, с практической точки зрения реализация и сопровождение на всех этапах жизни системы сталкиваются с некоторыми трудностями.

Наибольшей трудностью является именно сопровождение системы – в случае изменения требований к системе, качественно-количественных параметров ее функционирования, в том числе и к исходной аппаратной платформе, модификация управляющей среды крайне неэффективна и ресурсоемка.

Например, в случае появления дополнительного источника входных данных, включения в модель дополнительного регулятора, адаптировать цикл системы к новым условиям зачастую невозможно. Аналогично, при изменении производительности (скорости выполнения элементарных инструкций) цифрового блока необходимо вновь проводить комплексный пересчет параметров модели.

Естественно, при изменении архитектуры цифрового процессора, переходе на новый процессор, количество вносимых изменений заставляет отказаться от корректирования среды в пользу полной реконструкции. Становится гораздо выгоднее просто спроектировать еще одну систему, чем корректировать и вносить изменения в существующую.

Как правило, в таком случае совершают «откат» назад, к исходной математической модели, на ее основе строится новая среда – чаще всего с нуля. Результаты, полученные в ходе эксплуатации предыдущей версии программ и библиотек – в том числе практические наработки в виде исходных кодов программы – либо не используются, либо коэффициент вовлечения их в рефакторинг (реорганизацию) прототипа оказывается настолько малой, что ее можно не учитывать.

Подобная ситуация связана еще и с тем, что изначально отображение математических моделей управляемого объекта и регулятора в примитивы среды осуществляется на чрезвычайно низком уровне, максимально приближенном к аппаратным особенностям платформы (аппаратной организации, времени исполнения машинных команд). В качестве примера можно привести часто встречающуюся ситуацию организации программной задержки выполнения кода путем подсчета времени, требуемого на исполнение одной машинной инструкции либо некоторому их набору.

Отсутствие промежуточного уровня приводит к нежелательной «привязке» модели к фиксированной аппаратной конфигурации. Дополнительно этому способствуют преобладание использования языка программирования низкого уровня, чаще всего – языка ассемблера для конкретной платформы (по нашему мнению – зачастую необоснованное), а также, что немаловажно – низкая культура программирования и проектирования подобных программных средств [3].

Предлагаемый подход нацелен на использование новой модели построения управляющего программного обеспечения – ***событийно-ориентированной.*** В этом случае – с точки зрения разработки регулятора – объект управления и система управления (регулятор) рассматриваются как источник событий, так и их приемники. Адекватная реакция системы на события объекта обеспечивает корректное поведение объекта управления.

Под термином «событие объекта управления» мы будем понимать факт изменения состояния объекта управления, информация о котором (об изменении) может быть передана регулятору. Для каждого конкретного объекта управления можно определить множество событий и отношения между ними, такие, например, как правила появления и приоритет. Это множество будем называть ***«протоколом».***

Важным преимуществом подобного подхода является возможность применения понятия ***«контракт»*** к каждому объявленному протоколу, т.е. на этапе проектирования строго определить границы подмножества, в котором будут находиться события, актуальные для данного протокола. Реакцию регулятора также весьма желательно сводить в протокол. Однако в чистом виде такой прием является трудоемким, так как объект управления, как правило, обладает установившейся структурой, а воздействие на него регулятора не ограничивается исключительно информационными в их привычном понимании сигналами – чаще всего объект управления требует наличия «силовых» воздействий. Поэтому в цепь действия регулятора, которая передает события объекту управления, ставится промежуточное преобразовательное звено, которое теперь может быть описано с использованием понятий «событие» и «протокол событий».

Описание системы в терминах входных и выходных протоколов дает возможность однозначно спроектировать регулятор, который будет заведомо переносим, т.е. его структура не будет зависеть от структуры управляемого объекта, и – минимально – от конкретного вычислительного модуля, а только от используемых протоколов обмена событиями.

Вспомним, что и в классическом подходе существовало понятие асинхронной операции и «прерывания». Прерыванием называлось именно событие источника, информирующее приемник событий о смене состояния источника. При этом в зависимости от реализации приемник был вынужден приступить к выработке адекватного ответа на данное событие или самостоятельно принять решение о том, чтобы отложить выработку ответа либо вовсе игнорировать поступившую информацию.

Более того, раннее рассмотренная нами модель – модель с жестким командным циклом – также может быть представлена в терминах событийно-управляемой модели при следующих посылках: в системе присут-

ствует единственное событие – начало нового кванта времени (цикла модельного времени), реакцией на которое является обновление выходных величин регулятора. Протокол для него в минимальном варианте описывает периодичность возникновения этого события.

Итак, согласно данному подходу, вся информация распространяется в системе в виде событий, характеристики которых фиксированы соответствующими протоколами, также определенными разработчиком.

Теперь события как порции информации является самостоятельной сущностью. Вполне логичным оказывается следующий шаг, определяющий, что источники и приемники информации в системе также должны быть самостоятельными, строго определенными сущностями.

Изучение существующих сред [4] и анализ задач, для решения которых они предназначены, указывает на то, что в каждом случае можно выделить такие сущности – отдельные структурные элементы, на которых основана работа вычислительного модуля.

В подобной библиотеке должны присутствовать такие примитивы, как наиболее важные структуры данных и алгоритмы-процедуры для работы с ними. При этом правильность работы тем более легко доказать, чем более универсальным, компактным и легковесным будет эта обособленная библиотека – микроядро управляющей среды. Корректность работы ядра есть, на наш взгляд, наиболее важный параметр работы, ведь его отсутствие сводит на нет все остальные преимущества системы.

Даже состоящая из минимального количества примитивов, библиотека ядра представляет собой сложнейший комплекс данных и процедур. Эксплуатация и проверка ядра может потребовать замены некоторой части операционной библиотеки для получения дополнительной функциональности, адаптации к новым требованиям по быстродействию или аппаратным ресурсам. Это преимущество, очевидно, у библиотек, более организованных и предоставляющих более структурированный интерфейс. Для таких ядер адаптация к аппаратуре, включая доработку, отладку и тестирование, может быть выполнена максимально быстро.

Немаловажным фактором в пользу такого подхода является возможность модификации существующей системы, так как все примитивы системы являются стандартными, т.е. процедуры включения их в существующие системы или изъятия также являются стандартными.

Многочисленные примеры сред, а также теоретические исследования в этой области [5] указывают на то, что в составе операционной библиотеки подобного рода весьма желательны следующие компоненты, которые можно условно разбить на три группы:

- задачи как единицы исполнения (потоки/процессы исполнения в нашем случае);
- средства межзадачной коммуникации и синхронизации;
- средства системного и прикладного интерфейса;

Все эти элементы – примитивы – можно утилизировать, собрав в операционную библиотеку в виде исходных текстов или уже оттранслированного кода, обеспечив, таким образом, возможность их повторного использования. Эта мера, кроме того, гарантирует исправно работающую, т.е.

корректную реализацию данных элементов. Дальнейшее связывание этой библиотеки с кодом пользовательского приложения обеспечивает получение готовой управляющей среды, часть из которой ***заведомо корректна***. Кроме того, есть возможность значительно улучшить важный фактор – скорость разработки и, как следствие, затраты на неё.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Документы института системного программирования http://www.ispras.ru/.
2. *Фонин Ю., Грассман С.* Архитектура и принципы построения операционной среды «миниОС». Институт Системного Программирования РАН, 2004.
3. *Буч Г.* Объектно-ориентированный анализ и проектирование, 1999.
4. Документация по существующим операционным средам: AvrX (http://www.larrybarello.org), TinyOS (http://www.tinyOS.org), FreeRTOS (http://www.freeRTOS.org).
5. Разработка ОС реального времени для цифрового сигнального процессора http://www.citforum.ru/.

О.Н. Родзина, Л.С. Родзина

ЭВРИСТИЧЕСКОЕ ПРОГРАММИРОВАНИЕ И ОПТИМИЗАЦИЯ, ОСНОВАННАЯ НА ПРИРОДНЫХ АНАЛОГИЯХ

Модели эволюционной оптимизации являются далеко не единственными, построенными на природных аналогиях. Другими эвристическими процедурами являются метод отжига, пороговые и потоковые алгоритмы, метод рекордных оценок, конкурирующие с эволюционными вычислениями при решении задач практической оптимизации. Конкуренция происходит по критериям универсальности (пригодность для решения широкого класса задач оптимизации) и вычислительной мощности (качество полученного решения в сочетании с трудоемкостью его получения). Попробуем экспериментально установить сходство и различие этих методов с эволюционными алгоритмами.

Метод отжига был предложен С. Киркпатриком в начале 80-х годов для решения комбинаторных задач оптимизации. В качестве природного аналога был взят процесс кристаллизации эмалевой субстанции. В ходе ее охлаждения степень свободного движения молекул постепенно становится ограниченной, достигая точки равновесия с минимальным энергетическим уровнем в кристаллической структуре. Скорость ее охлаждения подчиняется распределению Больцмана. Аналогия с физическим процессом здесь заключается в том, что решения оптимизационной задачи соответствуют состояниям системы в процессе охлаждения физической субстанции; целевая функция соответствует ее энергетическому уровню; процедура поиска оптимального решения аналогична поиску состояния системы с минимальным энергетическим уровнем, а температура является параметром для управления процедурой оптимизации. Проведенное программное модели-