УДК 004.41 + 004.4'2

С. А. Белоконь, М. Н. Филиппов

Институт автоматики и электрометрии СО РАН
пр. Коптюга, 1, Новосибирск, 630090, Россия
E-mail: master1981@mail.ru

МЕТОД ПОСТРОЕНИЯ МНОГОПЛАТФОРМЕННОЙ ОТКРЫТОЙ МОДУЛЬНОЙ SCADA-СИСТЕМЫ

Представлены метод построения, архитектура и особенности реализации разработанной в ИАиЭ СО РАН SCADA-системы, в основу которой положены концепция модульности и использование открытых стандартов. Описан пример применения в автоматизированной системе диспетчерского управления движением поездов Новосибирского метрополитена.

Ключевые слова: открытые системы, SCADA-системы, динамический интерфейс, диспетчерское управление, метро.

Введение

В настоящее время подавляющее большинство SCADA-систем (Supervisory Control And Data Acquisition – диспетчерское управление и сбор данных) предназначено для использования исключительно под управлением операционной системы MS Windows, а буквально единицы оставшихся либо не составляют им конкуренции в плане функциональности, либо также написаны под какую-нибудь конкретную операционную систему и не могут быть многоплатформенными. Это является серьезным препятствием при разработке распределенных систем, в которых к некоторым узлам (например, осуществляющим управление) предъявляются повышенные требования по отказоустойчивости, и необходимо использование хорошо зарекомендовавших себя решений на базе QNX или Linux, а второстепенные узлы (например, реализующие только функции наблюдения) могут представлять собой обычные персональные компьютеры с более привычной для пользователей и обслуживающего персонала операционной системой MS Windows.

При разработке описываемой SCADA-системы помимо переносимости использованы и другие стандарты открытых систем, что на сегодняшний день не просто расценивается как преимущество, а является необходимым условием развития в сфере информационных технологий.

Поскольку различные категории специалистов и организаций зачастую трактуют термин «открытые системы» по-разному, отражая особенности собственной технической или коммерческой политики, сразу уточним, что в данной статье используется независимое и исчерпывающее определение, сформулированное комитетом IEEE POSIX 1003.0. В соответствии с этим определением открытой называется «система, реализующая открытые спецификации интерфейсов, служб и форматов данных, достаточные для того, чтобы обеспечить:

- возможность переноса (мобильность) прикладных систем, разработанных должным образом, с минимальными изменениями на широкий диапазон систем;
- совместную работу (интероперабельность) с другими прикладными системами на локальных и удаленных платформах;
- взаимодействие с пользователями в стиле, облегчающем последним переход от системы к системе (мобильность пользователей)» [1].

Ключевой момент в этом определении – использование термина «открытая спецификация», что в свою очередь определяется

© С. А. Белоконь, М. Н. Филиппов, 2008

как «общедоступная спецификация, которая поддерживается открытым, гласным согласительным процессом, направленным на постоянную адаптацию новой технологии, и соответствует стандартам» [1; 2].

Совокупность концепций переносимости, масштабируемости, интероперабельности и дружественности к пользователю обеспечивает преимущества для всех категорий специалистов, участвующих в процессе информатизации.

Для пользователя это:

- новые возможности сохранения сделанных вложений благодаря постепенному развитию функций и замены отдельных компонентов без кардинальной перестройки всей системы;
- отсутствие зависимости от одного поставщика аппаратных или программных средств, а также возможность выбора продуктов при условии соблюдения поставщиком соответствующих стандартов открытых систем;
- дружественность среды, в которой работает пользователь и мобильность персонала, а также возможность использования информационных ресурсов, имеющихся в других системах и организациях.

Проектировщик информационных систем имеет:

- возможность совместного использования разных аппаратных платформ и прикладных программ, реализованных в разных операционных системах;
- возможности использования готовых программных продуктов и информационных ресурсов.
- удобные средства инструментальных сред проектирования.

И, наконец, разработчик программных средств получает:

- повышение скорости разработки за счет повторного использования программ;
- развитые системы разработки, программирования и тестирования программного обеспечения;
- широкие возможности модульной организации программных комплексов благодаря стандартизации программных интерфейсов [1].

[1] *Кузнецов С.* Открытые системы, процессы стандартизации и профили стандартов. <http://www.citforum.ru/database/articles/art_19.shtml>

Несмотря на очевидные преимущества открытых стандартов, в области разработки SCADA-систем до сих пор преобладают системы закрытого типа. Логично предположить, что главным образом это обусловлено желанием коммерческих компаний получить дополнительную прибыль, вынуждая вместе с основной SCADA-системой покупать дополнительные продукты только своей разработки. С этой целью производители программного обеспечения сознательно ограничивают доступность технической документации на протоколы обмена и форматы данных, либо накладывают лицензионные ограничения на их использование. Все это не только увеличивает суммарную цену продукта, но и существенно ограничивает его функциональность, так как один, даже идеально спроектированный программный комплекс, но замкнутый сам на себя, не в состоянии удовлетворить потенциальные потребности всех пользователей.

После нескольких лет использования коммерческих SCADA-систем и с учетом их выявленных недостатков в лаборатории нечетких технологий Института автоматики и электрометрии СО РАН сформулированы предварительные спецификации и принято решение о разработке SCADA-системы, основанной на открытых стандартах. В систему заложен принцип модульности (построения из автономных элементов с простыми и согласованными структурными связями между ними) для обеспечения большей гибкости архитектуры, расширяемости системы и упрощенного повторного использования кода.

Работа была начата в рамках проекта по модернизации системы управления движением поездов Новосибирского метрополитена; к настоящему времени с помощью полученного инструмента разработаны программы, успешно использующиеся на станциях метро и в центре управления. При этом новая система позволила не только увеличить надежность программ (по сравнению использовавшейся до этого коммерческой средой разработки InTouch компании Wonderware [3]), но и значительно сократить время разработки, что на практике демонстрирует значительные преимущества заложенных в архитектуру принципов.

Например, программы, разработанные в InTouch, функционируют исключитель-

но под управлением операционной системы MS Windows, а анализ системных сбоев показывает, что надежность такого решения далека от идеальной, в то время как разработанная открытая система может быть скомпилирована под QNX или Linux (который в настоящее время и используется в качестве базовой платформы), что приводит к значительному повышению надежности всего решения. Модульность также увеличивает надежность системы, поскольку позволяет уменьшить количество потенциально содержащего ошибки исполняемого кода в памяти, к тому же аварийная ситуация, возникшая во время выполнения кода конкретного модуля, приводит только к его повторной загрузке и не распространяется на всю систему.

С другой стороны, поддержка открытых стандартов существенно сокращает время разработки приложений за счет возможности выбора программы, наиболее подходящей для решения возникшей задачи.

Например, поскольку все конфигурационные файлы системы, включая описания графических объектов, основаны на формате XML [2], обеспечена возможность выбора наиболее удобного графического редактора из огромного количества бесплатных приложений (*sodipodi* или *inkscape* для SVG-картинок [3], *draw* или *dia* для XML описания интерфейса пользователя, примеры см. ниже), и, что очень важно при коллективной разработке проекта (но, как и в большинстве других коммерческих SCADA-систем, отсутствует в InTouch), можно использовать систему управления версиями (CVS или SVN) для работы со всей конфигурацией системы, включая графическую информацию.

Открытый протокол общения между ядром и модулями позволяет легко расширять функциональность системы, выбирая более удобные для конкретной задачи решения. Например, экспериментальный блок мониторинга и перезапуска служб резервного компьютера написан на языке программирования Python и легко интегрируется в основную систему, разработанную на C++.

[2] Extensible Markup Language (XML).<http://www.w3.org/XML>

[3] Scalable Vector Graphics (SVG). XML Graphics for the Web. <http://www.w3.org/Graphics/SVG>

Архитектура системы

Внутренняя архитектура системы подчинена разработанной в рамках проекта концепции динамического интерфейса, основной идеей которого является *организация всех разделяемых между частями системы данных и способов взаимодействия в виде единой динамически изменяющейся структуры*. Эта структура (*DiBroker* – Dynamic Interface Broker) служит для хранения и наименования объектов (*DiObject*), каждый из которых описывает разделяемую переменную или метод модуля – одну точку интерфейса модуля.

С точки зрения модуля регистрация собственного интерфейса выглядит как операция создания нескольких файлов на своеобразной файловой системе, поскольку *DiBroker* поддерживает такие же функции создания, удаления и получения списка объектов, как и большинство других файловых систем, и объекты точно так же организованы в виде древовидной структуры с узлами-каталогами, причем в качестве имени-идентификатора выступает строка с символом-разделителем между элементами – путь от корневого каталога до объекта. Основным же отличием от обычной файловой системы является то, что *DiBroker* не хранит данные самостоятельно, он лишь служит для диспетчеризации разделяемых процедур и данных, находящихся внутри других модулей. Удаление объекта из *DiBroker* означает лишь то, что этот объект перестают видеть другие модули, т. е. он перестает быть разделяемым.

Разделяемые объекты представляют собой данные любого типа (например, *DiInt*, *DiString*, *DiTimer* или *MyType* – произвольный тип, определенный пользователем) с набором процедур для регистрации и оповещения подписчиков о запрашиваемых событиях. Например, модуль может «подписаться» на изменение выбранной переменной, т. е. зарегистрировать свою процедуру так, что она будет вызываться всякий раз при изменении этой переменной.

Адресация разделяемых объектов внутри модулей выполняется с помощью указателей с подсчетом ссылок (*SharedPtr<T>*) таким образом, чтобы объект не уничтожался до тех пор, пока на него ссылается хоть один модуль.

В качестве иллюстрации рассмотрим реализацию простой практической задачи: су-

ществует задаваемое в конфигурационном файле количество вольтметров, измеряющих напряжения в различных точках, и нужно включить реле звонка, если напряжение на одном из них ниже допустимой границы. Для решения необходим модуль **Вольтметр**, который общается с конкретным устройством через COM-порт, модуль **Логика** для сравнения напряжений с 220 В и модуль **Реле звонка**. Конфигурационный файл в формате для встроенного модуля *variables* может выглядеть так:

<variable name="измерения/вольтметр 1/конфигурация" type="string" value="COM1, 19200"/>

<module name="измерения/вольтметр 1" file="modules/stable/vm52.so" />

<variable name="измерения/вольтметр 2/конфигурация" type="string" value="COM2, 9600"/>

<module name="измерения/вольтметр 2" file="modules/stable/vm52.so" />,
что означает создание двух переменных (*"вольтметр 1/конфигурация"* и *"вольтметр 2/конфигурация"* в каталоге *"измерения"*) и загрузку двух одинаковых модулей (*"vm52.so"*) с домашними каталогами *"измерения/вольтметр 1"* и *"измерения/вольтметр 2"* соответственно (см. рис. 1).

При инициализации модуль **Вольтметр** считывает значение своей переменной *"конфигурация"* (*DiString config = DI_STRING(broker, "./конфигурация")*) и создает разделяемую переменную *"./напряжение"* (*DI_ATTACH(broker, "./напряжение", v)*), в которую периодически записывает получаемое напряжение (*v = buffer[7]*).

Затем подгружается модуль **Реле звонка** (*<module name="оповещение/звонок 1" file="modules/testing/relay.so"/>*), который регистрирует переменную *"оповещение/звонок 1"* (*DI_ATTACH(broker, "./", v)*) и обработчик на ее изменение (*DI_ATTACH_HANDLER(v, L::onChange)*), для включения/выключения звонка в зависимости от значения переменной (*v > 0 ? alarm_on() : alarm_off()*).

Далее в конфигурационном файле идут строки:

<variable name="логика 1/ввод" type="string" value="измерения/вольтметр \d+/напряжение"/>

<variable name="логика 1/минимальное значение" type="int" value="220"/>

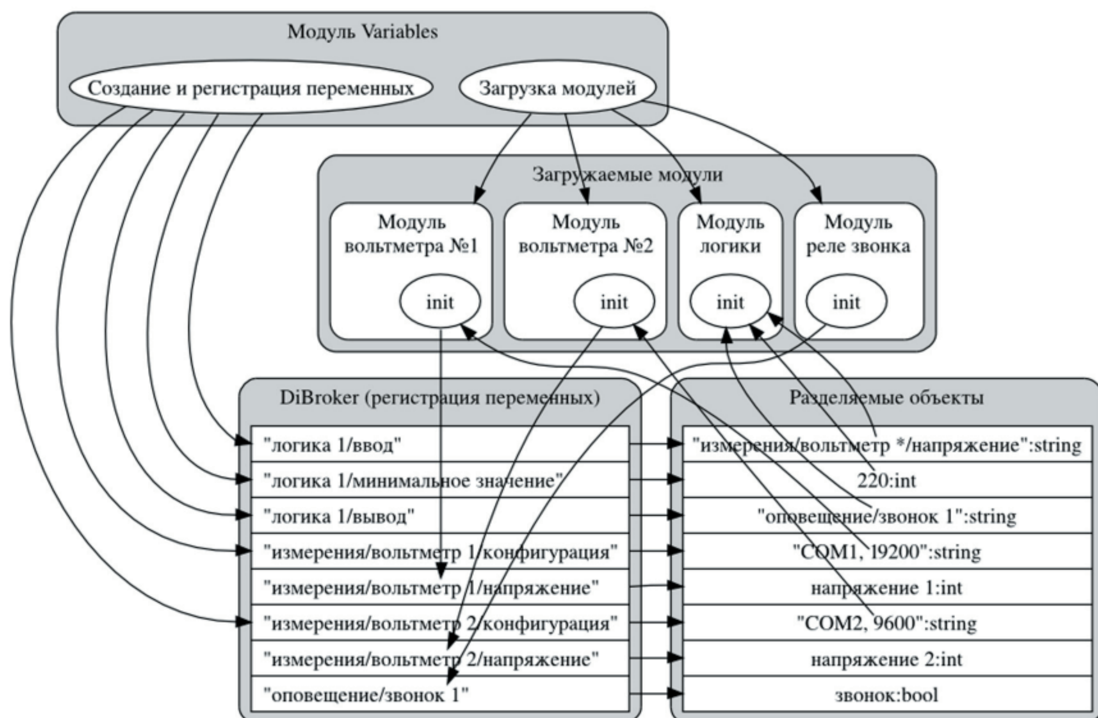

Рис. 1. Блок-схема инициализации системы контроля напряжения

<variable name="логика 1/вывод" type="string" value="оповещение/звонок 1"/>

<module name="логика 1/" file="modules/demo/logic1.so" />

При инициализации этот модуль, в соответствии с регулярным выражением (*"измерения/вольтметр \d+/напряжение")*, получает список всех переменных напряжения (*list = broker.ls(DI_STRING(broker, "./ввод"))*), устанавливает обработчик на их изменение (*DI_ATTACH_HANDLER(list[i], L::onTest)*), и, в случае падения напряжения, записывает *1* в переменную, имя которой хранится переменной *"./вывод"* (*limit = DI_INT(broker, "./максимальное значение"); ... if (list[i] < limit) DI_INT(broker, DI_STRING(broker, "./вывод")) = 1*).

В результате полный цикл включения звонка выглядит следующим образом (см. рис. 2): измеренное одним из вольтметров новое значение напряжения присваивается переменной *"измерения/вольтметр x/напряжение"*, это изменение приводит к вызову обработчика *onTest()* модуля **Логика**, и если проверка условия обнаруживает падение напряжения ниже допустимого минимального уровня, как следствие происходит изменение переменной *"оповещение/звонок 1"*, что в свою очередь приводит к вызову обработчика *onChange()* модуля **Реле звонка**, который и выдает необходимую команду на выполнение.

Несмотря на то, что этот пример намеренно усложнен для демонстрации системы взаимодействия между независимыми модулями только через разделяемые данные (без использования непосредственных вызовов функций модулей), вся реализация занимает около 20 строк кода и 10 строк конфигурации. При этом система поддерживает замену одних модулей другими во время исполнения и практически полностью может быть переконфигурирована минимальными изменениями.

Например, подключение еще одного вольтметра и изменение минимального значения выполняется без остановки работающей системы простой загрузкой модуля console и набором в нем соответствующих команд:

set "измерения/вольтметр 3/конфигурация" = "COM3, 9600"

load "измерения/вольтметр 3" modules/stable/vm52.so

set "логика 1/минимальное значение" = 100

Еще один пример: если подгрузить модуль view со следующей конфигурацией

<object type="svg" x="100" y="100" w="50" h="50"

variable="оповещение/звонок 1" onClick='оповещение/звонок 1=1'>

<state id="0" svg="rects_etc/rect_green"/>

<state id="invalid" svg="rects_etc/rect_red"/>

</object>,

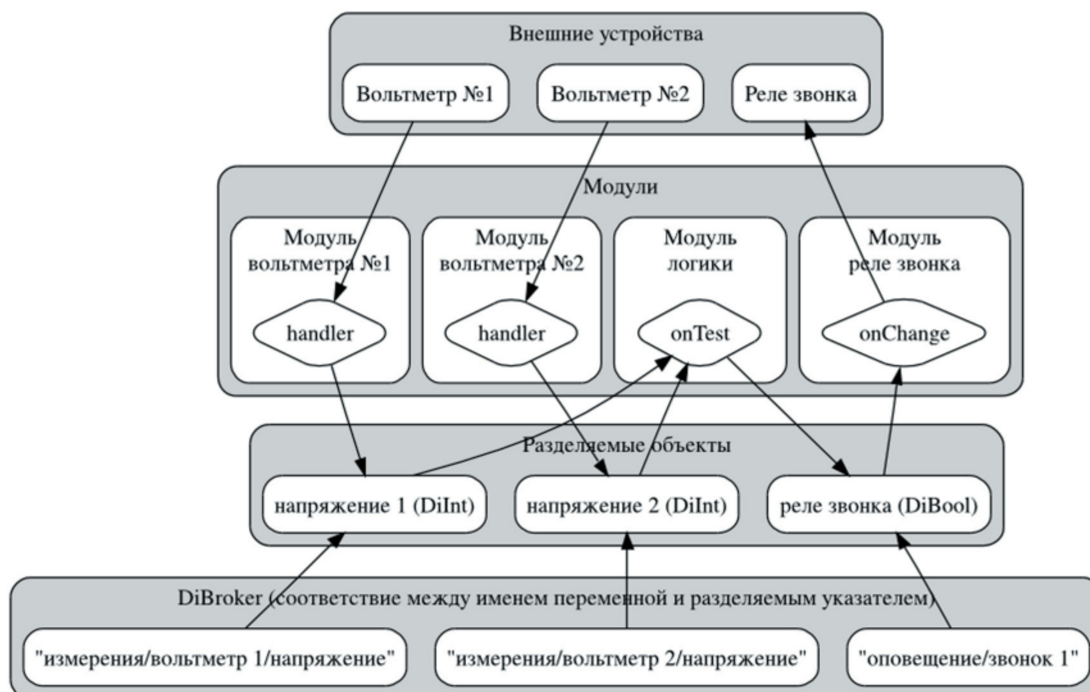

Рис 2. Блок-схема рабочего процесса системы контроля напряжения

то будет проинициализирована графическая система пользователя: в данном случае окно с квадратом, цвет которого зависит от состояния переменной «оповещение / звонок 1» (зеленый, если переменная равна нулю, и красный – в противном случае), а также, активировав этот квадрат манипулятором «мышь», можно включить реле звонка напрямую (более подробно о модуле ***view*** см. ниже).

Реализация

Приведенные примеры показывают, что ядро системы отвечает за очень ограниченный набор функций загрузки модулей и организацию взаимодействия между ними, поэтому реализация довольно компактна, к тому же выполнена в виде набора С++ шаблонов, что позволяет при разработке модулей не использовать никакие дополнительные библиотеки и интегрировать в систему пользовательские типы данных простым определением (например, *typedef DiType<int> DiInt*), см. иерархию классов, созданную системой документирования исходных текстов программы **doxygen**, изображенную на рис. 3.

В основу механизма оповещения об изменениях состояния положен шаблон проек-

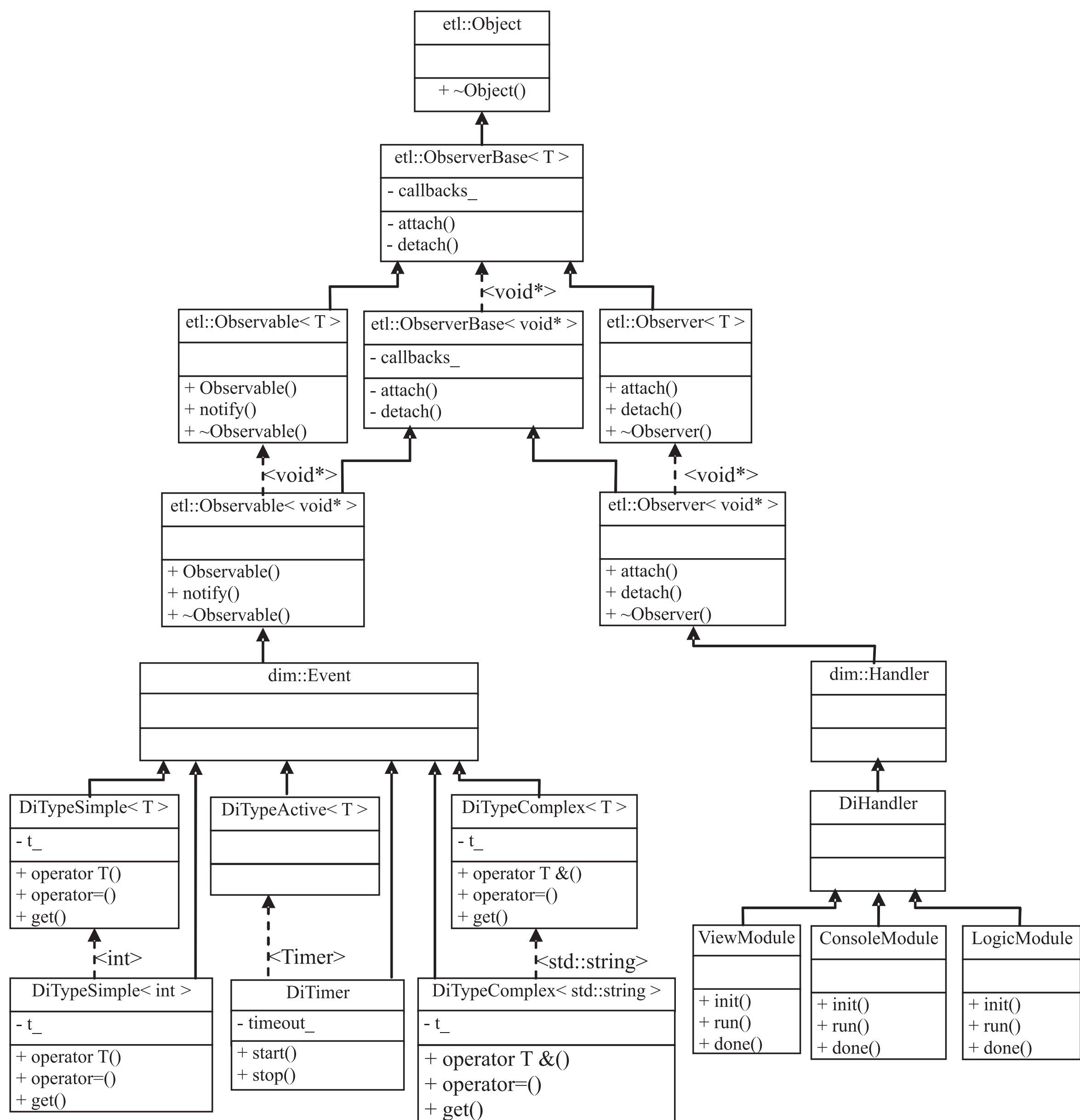

Рис. 3. Фрагмент UML-диаграммы, изображающий граф наследования ObserverBase, для типов DiInt, DiString, DiTimer и модулей ViewModule, ConsoleModule и LogicModule

тирования *Observer* («Наблюдатель») [4], реализованная модель которого построена на наследовании от базовых классов *Observer* и *Observable* с внутренними списками перекрестной регистрации, что позволяет получить:

- однотипность «делегата» (delegate – объект «обратного вызова») для непосредственного хранения в контейнерах (экономия ресурсов памяти);
- «прямой вызов» (direct call) без дополнительных разыменований указателя для типизированного вызова (оптимальная производительность);
- отсутствие опасных преобразований буфера указателя на функцию-член класса и автоматическое отключение от уничтоженного объекта (повышение безопасности).

Дополнительно реализован механизм предотвращения рекурсивных вызовов в случае появления циклов на графе зависимостей (например, цикл: переменная 1 изменяется в обработчике, установленном на изменение переменной 2, а переменная 2 изменяется в обработчике, установленном на изменение переменной 1).

Файловая система объектов организована таким образом, что позволяет избежать дополнительных расходов на поиск объектов по имени после инициализации модуля, поскольку работа с переменными происходит по прямым ссылкам; более того, когда модули уже получили ссылки на необходимые им разделяемые данные и зарегистрировали свои переменные, файловая система может быть полностью выгружена из памяти.

Скажем, *sum* – это ссылка на *DiInt*, проинициализированная как *DI_INT(broker, «демо/сумма/результат»)*, а *v1* и *v2* – ссылки, как проинициализированные *DI_INT(broker, «демо/сумма/переменная 1»)* и *DI_INT(broker, «демо/сумма/переменная 2»)* соответственно, тогда код *sum = v1 + v2* выполняется без накладных расходов (разумеется, время на организацию вызовов обработчиков изменения *sum* учитывается отдельно), точно так же, как если бы это были ссылки на локальные (неразделяемые) переменные.

Для достижения этого результата написан брокер запросов (*DiBroker*), который использован вместо системного *D-BUS*, использовавшегося в предыдущих версиях.

Модули представляют собой компилирующиеся независимо от основной программы (необходимо лишь включить заголовочный файл *«DI.hpp»*) разделяемые объектные файлы (*shared objects*), которые могут быть загружены и выгружены ядром системы во время исполнения.

Типичная схема поведения модуля – это получение ссылок на входные/выходные данные, регистрация обработчиков событий (например, *onTimer()*, чтобы периодически опрашивать устройство, *onChangeParameters()*, чтобы изменить параметры этого устройства) и запись своих результатов в выходные переменные при вызове этих обработчиков.

В процессе решения различных задач разработан ряд модулей специального назначения (*voltmetr, serial, modbus, proxysql* и т. п.) и несколько модулей общего назначения. Поскольку три из числа последних (*console, variables, view*) уже упоминались в примере выше и используются во всех эксплуатируемых системах, имеет смысл рассмотреть их более подробно.

Первый модуль (console) представляет собой пользовательский интерфейс, предоставляющий возможность в любой момент времени получить список переменных, посмотреть и изменить их значения, а также загрузить, выгрузить и переинициализировать любой модуль. Первоначально предназначавшийся для переконфигурации системы без остановки рабочего процесса, этот модуль активно используется разработчиками при отладке, позволяя моделировать желаемую ситуацию и выявить все нюансы поведения системы.

Следующий модуль (variables) служит для конфигурации системы: он обрабатывает переданный ему в качестве параметра XML-файл и инициализирует описанные там переменные и модули. Его функции были рассмотрены достаточно подробно в приведенном выше примере.

Модуль графического интерфейса пользователя (view) реализует широкий набор функций, рассмотрим лишь использованное в приведенном выше примере отображение объектов с заданным дискретным набором состояний. Конфигурационный файл модуля также имеет формат XML, в котором описываются отображаемые объекты разных типов (например, *«page», «label», «svg», «clock»*) и их характеристики (расположение на странице, размеры и т. п.). Эти объекты ассоциированы с переменными системы: модуль

устанавливает обработчики на изменение переменных и при изменении значения переменной изменяет соответствующие этой переменной графические объекты.

Графические объекты типа «*svg*», как видно из названия, для каждого описанного состояния отображают картинку, хранящуюся во внешнем файле формата SVG (Scalable Vector Graphic – масштабируемая векторная графика, стандарт W3C). Нужно отметить, что эти изображения компилируются при загрузке модуля в формат *pixbuf*, так что накладные расходы на чтение и разбор файлов во время исполнения отсутствуют.

Модуль *view* использует графическую библиотеку *GTK+*, которая обеспечивает современный интерфейс (интернационализацию, локализацию, специальные возможности) и существует на большинстве платформ (*UNIX, MS Windows, Mac OS X*) [4].

Поскольку конфигурационный файл модуля *view* также основан на открытом формате XML, обеспечена возможность разработки пользовательского интерфейса с использованием одной из множества свободно распространяемых XML-совместимых программ-редакторов.

Некоторое время казалось удобным расставлять и редактировать объекты на экране в графическом редакторе *Dia* (*open source editor for diagrams, graphs, charts* и т. д.), для чего его библиотека была расширена необходимыми элементами (см. рис. 4). В результате получался XML файл, который после довольно простого XSLT преобразования использовался как входной файл модуля *view* [5]. Но практика показала, что в большинстве случаев предпочтительнее автоматизировать этот процесс.

Например, на простую задачу расстановки на экране в виде таблицы ста одинаковых реле, имена которых указаны во внешнем файле («*1 ROMO; 2 RGKNO; ...*»), разработчик, пользующийся графическим редактором, не только тратит невероятное количество времени, но и допускает ошибки, а элементарный *shell* скрипт справляется с ней за доли секунды.

Применение

В качестве примера конкретного использования описываемой системы рассмотрим работу, выполнявшуюся последней в рамках проекта «Автоматизированная система диспетчерского управления движением поездов Новосибирского метрополитена».

В 2004–2005 гг. в Институте автоматики и электрометрии СО РАН с участием ведущих специалистов службы сигнализации и связи Новосибирского метрополитена завершена разработка автоматизированной системы управления движением поездов (АСДУ). В июне 2005 г. одновременно с открытием станции «Березовая Роща» начат ввод в эксплуатацию автоматизированной системы диспетчерского управления движением поездов на Дзержинской линии Новосибирского метрополитена, предусматривающий поэтапную модернизацию диспетчерской централизации на всем метрополитене. Общая структура системы представлена на рис. 5.

Автоматизированная система диспетчерского управления движением поездов представляет собой трехуровневую структуру, состоящую из комплекса автоматизированных рабочих мест на базе промышленных компьютеров и программируемых контроллеров, распределенных на значительном пространстве и работающих в темпе реального технологического процесса.

Верхний уровень системы включает в себя оборудование, расположенное на центральном посту управления:

- основное и резервное автоматизированные рабочие места (АРМ) поездного диспетчера;
- сервер АСДУ Дзержинской линии метро с глобальной базой данных;
- рабочее место дежурного инженера.

Средний уровень – рабочие места дежурных по постам централизации и рабочие места электромехаников СЦБ всех станций, входящих в АСДУ.

Нижний уровень – программируемый логический контроллер (ПЛК), выполняющий функции устройства сопряжения с объектом и реализующий алгоритмы управления напольным оборудованием.

Аппаратура первого и второго уровней объединена кольцевой системой передачи данных. Все АРМы центрального поста и станционные АРМы являются равноправными абонентами этой высокоскоростной Интернет-сети. Команды и известительная информация между диспетчерским постом

[4] < GTK+. The GIMP Toolkit. http://www.gtk.org>

и станциями передаются по двойному оптоволоконному кольцу с использованием стандартного коммуникационного протокола TCP/IP.

Автоматизированные рабочие места на каждой станции связаны с программируемым логическим контроллером посредством локальной станционной сети ModBus+ [6].

Разработанная SCADA-система использовалась при создании программного обеспечения АРМ дежурного инженера и АРМ электромеханика, основные функции которых:

- получение информации и отображение состояния технологических объектов на станциях линии (на станции) метрополитена в режиме реального времени;

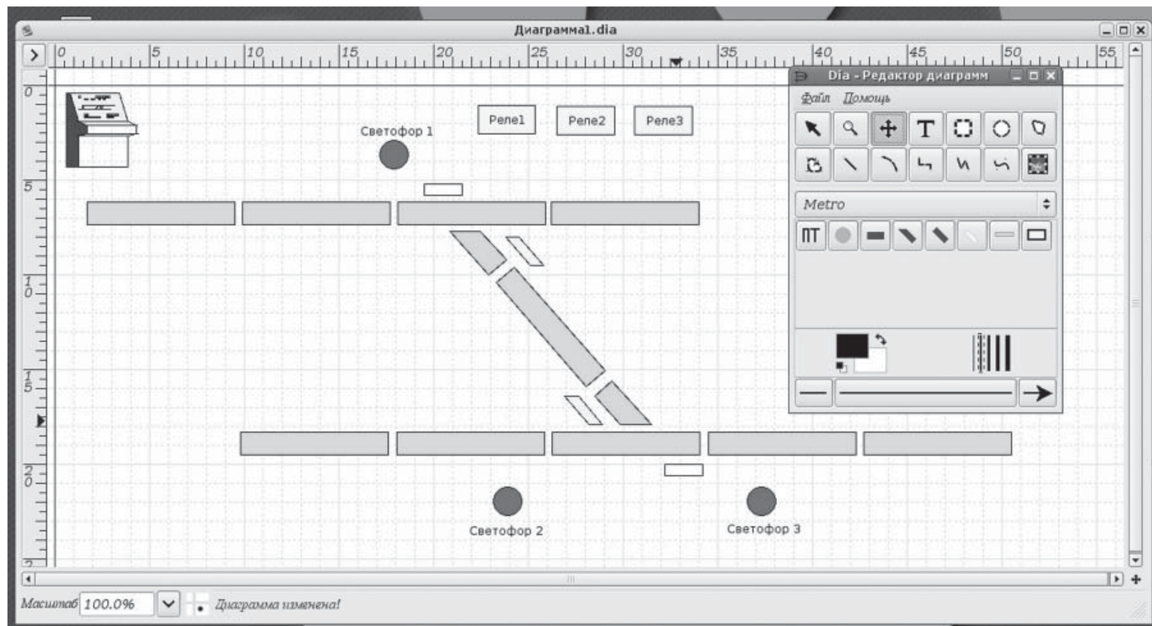

Рис. 4. Снимок экрана разработчика интерфейса пользователя, демонстрирующий использование редактора диаграмм *Dia*

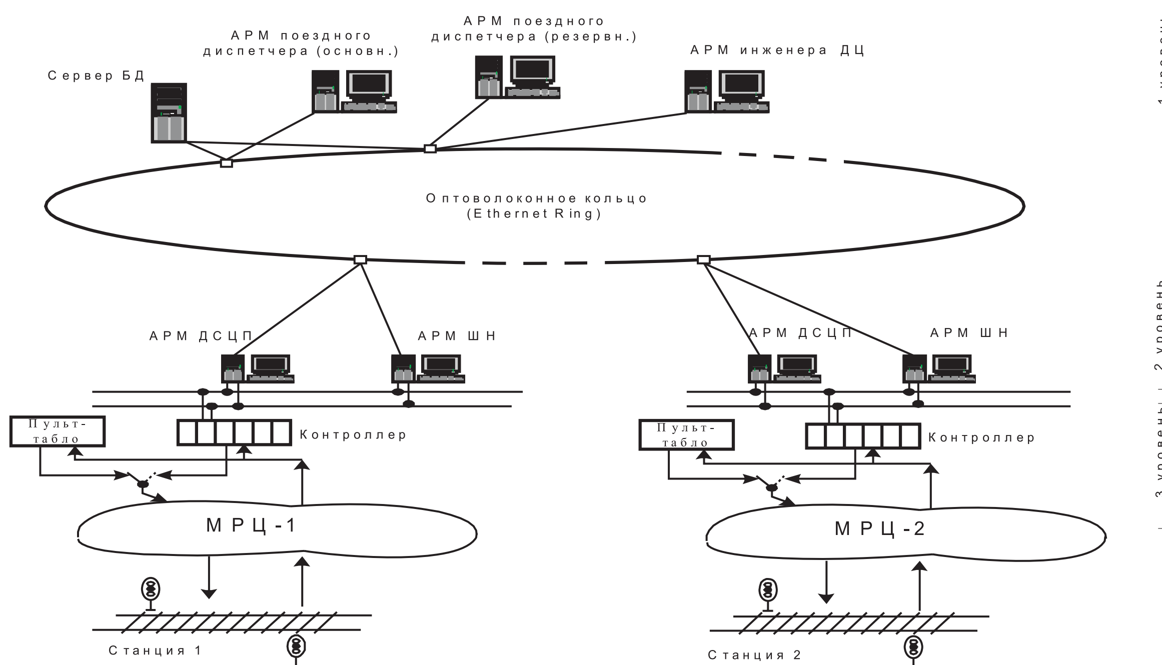

Рис. 5. Схема автоматизированной системы диспетчерского управления движением поездов

• формирование базы данных состояния технологических объектов на станциях линии (на станции) метрополитена;

• формирование и выдача отчетов;

• просмотр в динамическом режиме состояния устройств автоматики и поездной ситуации на любой станции линии (на станции) за любой период времени в любые из 30 предшествующих суток, включая день просмотра.

Стандартная конфигурация системы для АРМ электромеханика включает модули *variables, view, db* (работа с базой данных), *br_shn* (или *gm_shn* – логика, специфическая для данной станции) и несколько сотен переменных (например, 721 для станции «Березовая Роща» и 717 для «Пл. Гарина – Михайловского»). Конфигурация для АРМ дежурного инженера включает модули логики для всех подключенных станций.

В качестве примера рис. 6 демонстрирует видеокадр пользовательского интерфейса АРМ электромеханика: на трех страницах («Березовая Роща. Главное окно», «Второй путь» и «Контроль параметров») отображаются происходящие в реальном времени либо сохраненные в базе данных события, такие как изменения состояния оборудования (реле, сигналов, рельсовых цепей и т. п.) и действия дежурного по станции.

Управление программой максимально упрощено и интуитивно понятно: можно наблюдать за текущими изменениями (кнопка *«Воспр.»*), кнопками *«Назад» / «Далее»* или вводом даты выбрать интересующее событие, включить/выключить быструю перемотку, просмотреть журнал действий диспетчера и т. п.

Программа работает под управлением операционной системы Linux, которая настроена на выполнение сразу после загрузки единственного приложения с правами пользователя, не допускающими изменений в системе.

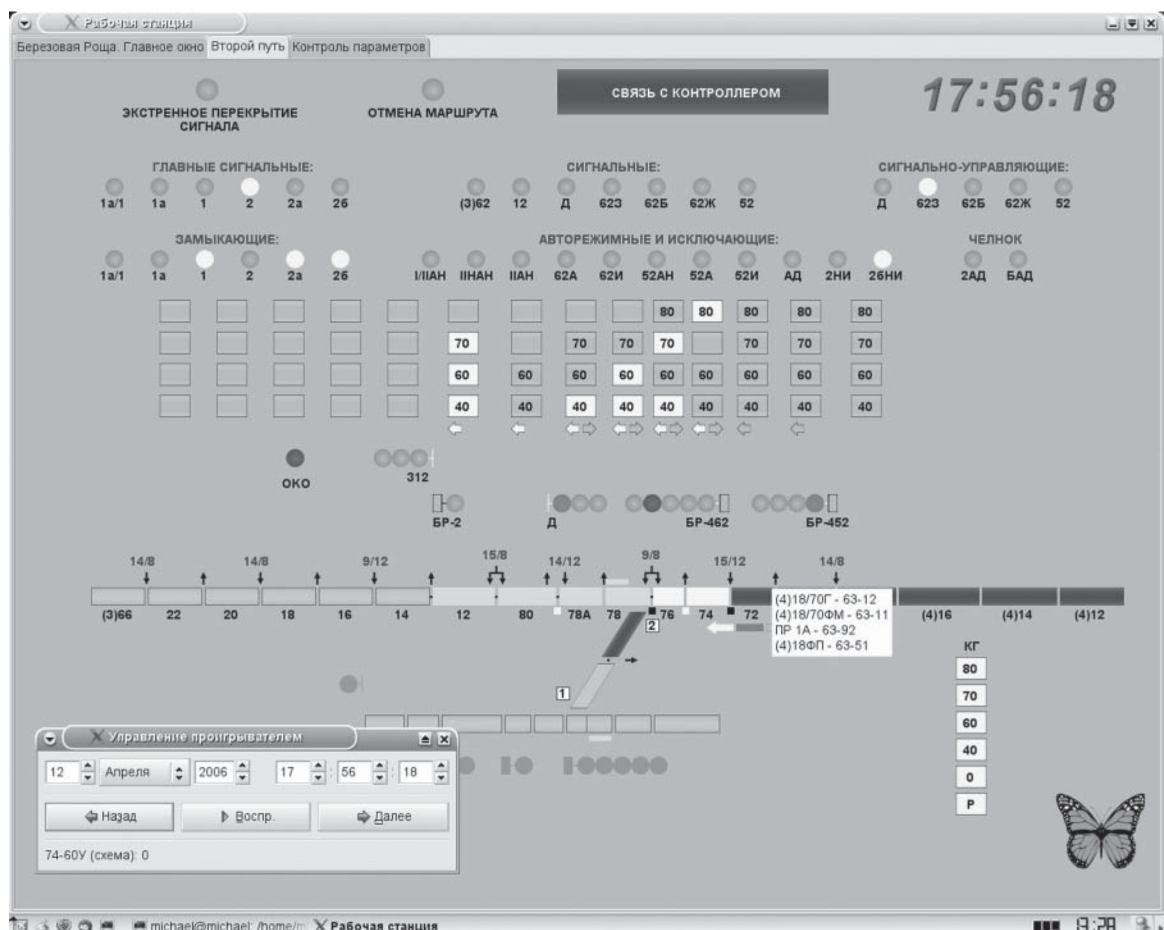

Рис. 6. Снимок экрана пользователя (АРМ электромеханика)

Общий размер программы, модулей, сообщений, конфигурационных и графических файлов составляет 1–1,5 MB дискового пространства; во время исполнения программа занимает менее 20 MB оперативной памяти. Предусмотрены также конфигурации для бездисковых машин с загрузочным CD или USB диском.

В описанной конфигурации АРМ электромеханика успешно работает с июня 2005 г.

Заключение

Предложен метод построения SCADA-системы, основными отличительными признаками которого являются специально разработанная архитектура *динамического интерфейса*, а также использование открытых стандартов на всех этапах создания.

На основе предложенного метода реализована многоплатформенная модульная SCADA-система. К настоящему времени с помощью полученного инструмента создано программное обеспечение, использующееся на станциях метро и в центре управления Новосибирского метрополитена.

При этом новая система позволила не только увеличить надежность программ, но и значительно сократить время разработки.

Более чем двухгодичный опыт эксплуатации системы подтвердил эффективность решений, принятых при ее создании.

Ограничения предложенного метода и текущей реализации планируется исправить на следующем этапе развития системы, в частности, добавить средства безопасного программирования (такие как автоматическая верификация кода, разграничение и контроль доступа для подпрограмм, независимые адресные пространства драйверов и модулей и т. д.), а также расширить базу поддерживаемых устройств.

Список литературы

1. *Issak J., Lewis K., Thomson K., Strayb R.* Open System Handbook. A Guide to Building Open System. Publiched by THE IEEE STANDARDS PRESS The Institute of Electrical and Electronics Engineers, Ins. 1998 г.

2. *Открытые* системы. Материалы к программе развития и применения открытых систем в Российской Федерации. Казань, 1994.

3. *Средства* организации верхнего уровня системы автоматизации. SCADA-система InTouch. Модульные системы Торнадо. 2004.

4. *Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж.* Приемы объектно-ориентированного проектирования. Паттерны проектирования. СПб.: Питер, 2001.

5. *Белоконь С. А., Филиппов М. Н.* Открытая SCADA-система: разработка и реализация прототипа // Материалы науч.-практ. конф. молодых ученых и студентов НГУ и ИАиЭ СО РАН «Информационно-вычислительные системы анализа и синтеза изображений», 19–21 сентября 2006 г. Новосибирск, 2006. С. 45–48.

6. *Абрамов А. И., Белоконь С. А., Васильев В. В. и др.* Модернизация системы диспетчерского управления движением поездов метрополитена // Тр. VIII Междунар. конф. «Проблемы управления и моделирования в сложных системах», 24–28 июня 2006 г. Самара, 2006. С. 269–273.

Материал поступил в редколлегию 13.12.2007