СПЕЦИФИКАЦИЯ СРЕДСТВАМИ ЯЗЫКА XML СИСТЕМЫ ИНТЕРЕЙСОВ В ПРИЛОЖЕНИЯХ РЕАЛЬНОГО ВРЕМЕНИ

НИКИФОРОВ В. В., ШКИРТИЛЬ В.И.

УДК 681.3

Никифоров В.В., Шкиртиль В.И. **Спецификация средствами языка XML системы интерфейсов в приложениях реального времени.**

Аннотация. Изложен подход к использованию средств языка XML для спецификации межзадачных и внешних интерфейсов в программных приложениях реального времени. Спецификации позволяют представлять множество задач, составляющих программное приложение; последовательность сегментов кода в каждой из задач; множество интерфейсных элементов, обеспечивающих межзадачные связи; множество датчиков и эффекторов, реализующих связь программного приложения с внешним оборудованием. Элементы спецификации снабжаются динамическими характеристиками исполнения моделируемых объектов. Обработка спецификаций позволяет проверять гарантии своевременности выполнения функций системы реального времени.

Ключевые слова: системы реального времени, межзадачные интерфейсы, XML-модели, динамическая корректность программных приложений.

Nikiforov V.V., Shkirtil V.I. **Specification of interfaces in real-time software applications by XML forms.**

Abstract. An approach is expouned for XML use to specify intertask and external interfaces of real-time software applications. Specification of software application interfaces presents a set of tasks makes up software application; a set of interface elements, that implement intertask links; a sequence of code segments in each task; a set of sensors and effectors that link software application with external environment. Specification elevents equipted by dynamic characteristics of modeling objects.The specification processing permits to check feasibility of specified system.

Keywords: real-time systems, task interfaces, XML-models, dynamic correctness of software applications.

1. Введение. Построение рациональной и корректной системы интерфейсов программных приложений относится к ряду важных конструктивных решений при разработке архитектуры систем реального времени (СРВ). На протяжении десятилетий развития технологии программирования разработчикам предлагались различные средства спецификации интерфейсов в программных системах — от всевозможных модификаций сетей Петри [1] до UML-диаграмм [2]. Предлагавшиеся средства в основном ориентированы на проверку логической корректности программных систем путем визуального, а также в той или иной мере автоматизированного анализа.

Для систем реального времени необходимо обеспечивать не только логическую, но и динамическую корректность программного ком-

плекса, обеспечивать выполнение функций системы в заданных временных рамках. Проверка гарантий динамической корректности моделей программных приложений СРВ выполняется в два этапа: 1) модели, ориентированные на визуальное восприятие, представляются в аналитическом виде; 2) для автоматической обработки модели переводятся коды универсального языка программирования. Авторами разработан подход к представлению спецификаций системы интерфейсов средствами расширяемого языка разметки XML [3], позволяющий объединить эти два этапа. Средства XML могут быть использованы для построения таких формализованных моделей интерфейсов, которые в равной мере пригодны как для восприятия специалистами, выполняющими разработку, модификацию, сопровождение СРВ, так и для обработки автоматическими анализаторами, обеспечивающими, в частности, анализ выполнимости приложений, адекватных представляемым моделям.

Определяющее свойство программных приложений реального времени состоит в том, что они работают «в структуре времени, определяемой ходом внешних процессов». Требование соответствия между ходом исполнения программных компонент СРВ и ходом внешних процессов обусловлено наличием более или менее жестких временных рамок для информационных обменов с внешними процессами.

Требование соответствия между «структурой времени», отражающей ход внешних процессов, и «структурой времени» исполнения программных компонент является одним из проявлений общего требования к архитектуре рационально построенной СРВ — соблюдения *принципа структурного соответствия*. В самом общем виде этот принцип формулируется так: для рационально построенной СРВ структура комплекса программных объектов должна представлять собой преломленное отражение структуры множества внешних объектов [4].

Для СРВ характерно широкое разнообразие автоматизируемых узлов и агрегатов, их взаимосвязей, различий в темпах их функционирования. В связи с этим программные приложения реального времени строят в виде множества относительно самостоятельных, но взаимосвязанных программных компонент. Такие компоненты указанных приложений называют *задачами*. В ходе работы системы отдельная задача может исполняться многократно. Отдельное исполнение задачи называют *заданием*.

В состав интерфейсов программных приложений СРВ входят интерфейсы, связывающие задачи между собой и интерфейсы, связывающие задачи с внешними объектами. Поэтому в спецификации ин-

терфейсов программных приложений должны быть представлены следующие элементы:

— состав и характеристики внешних объектов;

— состав и характеристики тех интерфейсных объектов, которые обеспечивают реализацию межзадачных связей;

— связи программного приложения с внешними объектами;

— состав и характеристики задач с особенностями размещения в коде этих задач операторов доступа к интерфейсным объектам.

2. Моделирование внешних объектов СРВ. Для представления состава внешнего оборудования СВР средствами XML следует задать соответствующие типы элементов в спецификациях DTD (Document Type Definition — определении типа документа). Такие элементы предназначается для описания всех датчиков, всех эффекторов, через которые программное приложение взаимодействует с внешней средой:

```
<!ELEMENT environment (source+, effector+)>
```

2.1. Представление состава датчиков. Модель внешнего датчика содержит от трех до четырех атрибутов:

```
<!ELEMENT source>
<!ATTLIST source
    isr_p      IDREF
    name       ID              #REQUIERED
    periodic   ("yes"|"no")    #REQUIERED
    interval   #PCDATA         #REQUIERED>
```

Различают две разновидности датчиков — сигнальные и пассивные. Срабатывание сигнального датчика приводит к возникновению прерывания — порождению задания, выполняющего задачу по обработке данного прерывания (этот обработчик прерываний представляется указателем `isr_p` на элемент типа `isr`). Срабатывание пассивного датчика состоит в генерации цифрового кода, выставляемого на интерфейсном регистре соответствующего периферийного блока компьютерной системы. Такие коды считываются по инициативе прикладных задач — либо обработчиков прерываний, либо программно активизируемых задач (потоков). В описании XML-элемента, соответствующего пассивному датчику, указатель `isr_p` отсутствует.

Атрибут `interval` задает минимальную длину временного интервала между соседними срабатываниями датчика. Атрибут `periodic` используется для указания, возможно ли разнесение соседних срабатываний датчика на интервалы, превышающие значение `interval`. В случае датчиков, срабатывающих строго периодически, для атрибута `periodic` устанавливается значение "`yes`", в противном случае —

значение ”no”. Атрибут `name` используется для установления связей между внешними объектами и прикладными задачами СРВ.

2.2. Представление состава эффекторов. Модель внешнего объекта типа эффектора задается значениями трех атрибутов:

```
<!ELEMENT effector >
<!ATTLIST effector
    name          ID       #REQUIERED
    start_source  IDREF
    deadline      #PCDATA>.
```

Атрибут `name` служит для указания связи с прикладной задачей, выдающей управляющее воздействие на данный эффектор. Атрибут `start_source` указывает датчик, срабатыванием которого начинается очередной цикл вычисления управляющего воздействия, выдаваемого на данный эффектор. Атрибут `deadline` задает максимальное значение допустимой продолжительности цикла вычислений выдаваемого управляющего воздействия.

2.3. Пример состава внешних объектов СРВ. Представим себе простейшую автономную (без пульта оператора) СРВ, содержащую три датчика: 1) положения, 2) скорости и 3) тактов реального времени . Первый указывает положение исполнительного механизма, перемещаемого по единственной координате (например, по рельсу). Второй указывает скорость этого перемещения. Третий (таймер) генерирует импульсы прерываний с интервалом T единиц времени. Единственный эффектор принимает от компьютерной системы управляющее воздействие, влияющее на скорость движения исполнительного механизма.

Состав внешних объектов такой СРВ представляется следующим фрагментом XML-документа:

```
<environment>
     <!-- Пассивный датчик положения -->
     <source name=”position”
             periodic=”no”
             interval=”10”/>
     <!-- Пассивный датчик скорости -->
     <source name=”speed”
             periodic=”no”
             interval=”100”/>
     <!-- Единственный сигнальный датчик
                       – таймер РВ  -->
     <source name=”rt_timer”
             isr_p=”rt_isr”
```

```
            periodic="yes"
            interval="500"/>
    <!-- Эффектор, принимающий
         управляющие воздействия -->
    <effector name="force"
              periodic="no"
                       start_source="rt_timer"
              deadline ="300"/>
</environment>
```

Приведенный фрагмент XML-документа содержит указатель `isr_p` на компоненту модели программного приложения — задачу (обработчик прерываний), реагирующую на срабатывание сигнального датчика.

3. Модель программного приложения. Различаются два рода объектов, составляющих программное приложение СРВ: 1) *задачи*, реализующие требуемые алгоритмы функционирования приложения и 2) *интерфейсные объекты*, связывающие задачи в единое программное приложение. Доступ задач к интерфейсным объектам осуществляется через интерфейсные операции.

3.1. Интерфейсные операции. Точки доступа к интерфейсным объектам оформляются в коде программы операторами передачи/приема информационных посылок. Совокупность взаимосвязей задач, образуемая точками доступа к интерфейсным объектам, вместе с соответствующими механизмами составляют множество межзадачных интерфейсов.

Программный код каждой прикладной задачи состоит из последовательности операций. *Локальные* операции выполняют модификации элементов локального контекста задачи; результаты таких (локальных) операций непосредственно доступны только в рамках самой задачи, выполняющей локальную операцию. *Интерфейсные* операции, обеспечивающие межзадачные взаимосвязи и взаимосвязи программного приложения с внешним оборудованием, в свою очередь подразделяются на производящие и потребляющие. *Производящие* операции генерируют информационные посылки и направляют их (через интерфейсные объекты) другим активным заданиям или (через интерфейсные регистры периферийных блоков) на внешнее оборудование. При исполнении *потребляющей* операции информационные посылки извлекаются из интерфейсных объектов и используются в ходе дальнейшей работы. Потребляющая операция может повлечь приостановку текущего исполнения задачи. Если происходит обращение к интерфейсному объекту, в который еще не поступила запраши-

ваемая информационная посылка, то происходит приостановка исполнения текущей задачи (перевод в состояние ожидания поступления такой посылки).

При практической реализации различных СРВ используется большое разнообразие типов интерфейсных объектов (глобальных структур данных, очередей сообщений, сигналов, семафоров и т. п.). В рамках настоящей работы представлены две разновидности интерфейсных объектов: 1) мютексы (`mutex`) и 2) очереди сообщений (`queue`).

3.2. Мьютексы. Механизм доступа к интерфейсным элементам типа мьютексов является одним из наиболее широко используемых механизмов реализации межзадачных связей. Назначение мьютекса ~~состоит~~ в том, чтобы исключить одновременный доступ к ресурсу со стороны двух и более заданий.

Для обеспечения такого контроля перед входом в блок программы, взаимодействующий с ресурсом g (перед входом в критический интервал по ресурсу g) над соответствующим мьютексом выполняется операция «захват» (`lock` — потребляющая операция над мьютексом). Если к моменту исполнения операции `lock` мьютекс свободен, то он переводится в состояние «занят», текущее задание входит в охраняемый мьютексом критический интервал. Если задание (поток) пытается выполнить операцию «захват» над занятым мьютексом, то оно переводится в состояние «ожидание» момента освобождения этого мьютекса.

При выходе из критического интервала текущее задание выполняет над этим мьютексом операцию «освобождение» (`unlock` — производящая операция над мьютексом). Если в момент обращения к операции `unlock` отсутствуют потоки, ожидающие освобождения этого мьютекса, то мьютекс переводится в состояние «свободен». В случае наличия потоков, ожидающих освобождения этого мьютекса, один из этих потоков захватывает ресурс (входит в критический интервал); мьютекс остается в состоянии «занят».

Для представления мьютексов в XML-модели строятся элементы, соответствующие следующему формату:

```
<!ELEMENT mutex>
<!ATTLIST mutex name ID #REQUIERED>
```

Единственный атрибут `name` служит для построения указателей, связывающих модели задач с моделями используемых ими мьютексов.

Задачи, взаимосвязанные через доступ к мьютексам, перестают быть независимыми. Задача называется *независимой*, если соответст-

вующие ей задания не попадают в состояния ожидания системных событий, генерируемых другими заданиями. Задачи, связанные через мьютексы, становятся взаимозависимыми, поскольку при выполнении операции «захват» соответствующие им задания могут попасть в состояние ожидания момента освобождения требуемого мьютекса.

3.3. Очереди сообщений. Для очередей сообщений производящей операцией является операция засылки сообщения в очередь, а потребляющей — операция извлечения сообщения из очереди. Если очередь пуста, то задание, пытающееся получить сообщение из этой очереди, приостанавливается (до того момента, когда ему будет предоставлено новое засылаемое в эту очередь сообщение).

Очереди сообщений моделируются XML-элементами следующего вида:

```
<!ELEMENT queue>
<!ATTLIST queue name  ID       #REQUIERED
                size  #PCDATA  #REQUIERED>
```

Атрибут `name` служит для построения указателей, связывающих модели задач с моделями используемых ими очередей сообщений.

Если задача связана с очередью сообщений через потребляющую операцию, то соответствующее ей задание может попадать в состояние ожидания поступления очередного сообщения в эту очередь (следовательно, задача не является независимой). Задачи, связанные с очередями сообщений только через производящие операции и не связанные с мьютексами, являются независимыми задачами (соответствующие им задания не могут попасть в состояние ожидания).

3.4. Сегменты задач. Условимся рассматривать такие задачи, в которых интерфейсные операции не встречаются внутри операторов ветвлений и циклов. Тогда интерфейсные операции разбивают программный код задачи на последовательность сегментов. Первая интерфейсная операция в теле задачи завершает первый сегмент задачи, последующие операторы до второй интерфейсной операции включительно составляют второй сегмент задачи и так далее. В качестве параметра интерфейсной операции следующие четыре указателя:

1) мьютекса,

2) очереди,

3) интерфейсного регистра периферийного блока системы,

4) программно-активизируемой задачи.

В четвертом случае в результате выполнения синхронизирующего оператора порождается поток — экземпляр активизируемой задачи.

Последний сегмент программного кода задачи замыкается либо интерфейсной, либо локальной операцией.

XML-элемент, представляющий сегмент задачи, имеет от одного до трех атрибутов:

```
<!ELEMENT segment>
<!ATTLIST segment length #PCDATA  #REQUIERED
                  interface IDREF
                  op_type ("get"|"put")>
```

Значением атрибута `length` указывается объем процессорного времени, требуемый для выполнения сегмента. Атрибут `interface` предусмотрен для указания интерфейсного элемента, с которым связана интерфейсная операция, замыкающая сегмент.

Если последний сегмент задачи замыкается локальной операцией, то в спецификации этого сегмента атрибуты `interface` и `op_type` отсутствуют.

Если значением атрибута `interface` является задача (элемент типа `thread`), то выполнение замыкающей сегмент операции состоит в активизации этой задачи (порождении задания — очередного ее экземпляра задачи); атрибут `op_type` отсутствует.

Для интерфейсных регистров выполнение потребляющей операции означает считывание показания интерфейсного регистра, выполнение производящей операции означает засылку кода в интерфейсный регистр.

В спецификациях сегментов, замыкаемых операциями с мьютексами, синхронизирующими очередями, интерфейсными регистрами значение `get` атрибута `op_type` означает, что операция является потребляющей, значение `put` означает, что операция является производящей.

Приостановка текущего задания возможна только в конце таких сегментов, которые замыкаются потребляющими операциями над синхронизирующими элементами одного из двух типов — мьютексами или синхронизирующими очередями сообщений.

3.5. Обработчики прерываний. Для моделирования обработчиков прерываний используются XML-элементы следующего вида:

```
<!ELEMENT isr (segment+)>
<!ATTLIST isr name       ID      #REQUIERED
              prio_level #PCDATA >
```

Содержание элемента типа `isr` составляет один или несколько XML-элементов типа `segment`. Любой обработчик внешних прерыва-

ний должен быть независимой задачей, поэтому на значения атрибутов составляющих `isr` сегментов накладываются следующие ограничения:

— значением атрибута `interface` не может быть указатель мьютекса,

— если значением атрибута `interface` является указатель синхронизирующей очереди, то атрибут `op_type` обязательно должен иметь значение `get`.

Еще одно ограничение касается связей обработчика прерываний с внешними объектами: если значением атрибута `interface` является XML-элемент, представляющий датчик, то это должен быть пассивный датчик (т. е. в спецификации этого XML-элемента должен отсутствовать атрибут `isr_p`).

Атрибут `name` используется для указания связи с XML-элементом, представляющим сигнальный датчик — источник прерываний, обрабатываемых специфицируемой задачей.

Атрибут `prio_level` предусмотрен для указания аппаратно поддерживаемого уровня приоритета процессора, соответствующего специфицируемому обработчику прерываний. Атрибут `prio_level` требуется в случае, если компьютерная система поддерживает несколько уровней прерываний, в противном случае все обработчики прерываний равноправны, указание значения `prio_level` не требуется.

3.6. Программно-активизируемые задачи. В отличие от обработчиков прерываний, для которых соответствующие задания порождаются в результате срабатывания сигнальных датчиков, экземпляры программно-активизируемых задач порождаются в результате исполнения интерфейсных операций, замыкающих сегменты, для которых значением атрибута `interface` является указатель XML-элемента следующего типа:

```
<!ELEMENT thread (segment+)>
<!ATTLIST thread name   ID      #REQUIERED
                 prio   #PCDATA #REQUIERED>
```

Атрибут `name` используется при описании сегментов, у которых значением атрибута `interface` является задача. Атрибут `prio` предусмотрен для указания программно поддерживаемого приоритета, соответствующего специфицируемой задаче.

3.7. Программное приложение. В состав программного приложения СРВ входят: один или более обработчик прерываний, множество (возможно, пустое) программно-активизируемых задач и множество

(возможно, пустое) интерфейсных объектов (мьютексов и очередей сообщений):

```
<!ELEMENT application (isr+, thread*,
                      mutex*, queue*)>
<!ATTLIST application protocol ("PIP"|
                                "PCP"|"PCIP")>
```

Спецификация программного приложения, содержащего мьютексы, должна задавать значение атрибута `protocol`, уточняющего способ доступа к критическим интервалам, охраняемым мьютексами: протокол наследования приоритетов (priority inheritance protocol — `PIP`), протокол пороговых приоритетов (priority ceiling protocol — `PCP`), протокол превентивного наследования приоритетов (priority ceiling inheritance protocol — `PCIP`). Механизмы реализации этих протоколов представлены, например, в работе [5]. От выбора протокола доступа к разделяемым ресурсам зависит метод учета продолжительности критических интервалов при оценке времени отклика задач.

Для анализа актуальных свойств СРВ модель приложения должна рассматриваться совместно с совокупностью моделей внешних объектов, содержащих:

— данные о темпе внешних процессов (представляемых срабатываниями датчиков),

— допустимое время реакции программного приложения на внешние события (продолжительности временных интервалов между срабатываниям датчиков и выдачи соответствующих управляющих воздействий).

То есть для анализа свойств СРВ необходимо рассматривать полную модель системы, включающую в себя модель внешнего оборудования и модель программного приложения.

4. Полная модель СРВ. Формат XML-документа для задания полной модели СРВ имеет вид:

```
<!ELEMENT rt_system (environment,
                     application)>
```

Ниже приведены два примера моделей СРВ, в которых внешние объекты соответствуют спецификациям подраздела 2.3, а программное приложение реализуется различными способами.

4.1. Однозадачное программное приложение. Полная модель простейшей автономной СРВ, рассмотренной в подразделе 2.3, может иметь следующую структуру:

```
<rt_system>
    <environment>
            ... Описание состава внешних объектов
            ... из подраздела 2.3
    </environment>
    <application>
        <isr name="rt_isr">
            <segment length="10"
                     interface="position"
                     op_type="get">
            </segment >
            <segment length="10"
                     interface="speed"
                     op_type="get">
            </segment >
            <segment length="130"
                     interface="force"
                     op_type="put">
            </segment >
        </isr>
    </application>
<rt_system>.
```

Из определения элемента `thread` следует, что программное приложение состоит из единственной задачи — обработчика прерываний `rt_isr`. Задача `rt_isr` активизируется аппаратным таймером `rt_timer` каждые 500 единиц времени, состоит из трех сегментов:

— в первом считывается показание датчика положения (на это расходуется не более 10 единиц процессорного времени);

— во втором — показание датчика скорости (расходуется также не более 10 единиц);

— в рамках третьего выполняется вычисление управляющего воздействия, на эффектор выдается полученный результат (что занимает не более 130 единиц процессорного времени).

Задача `rt_isr` должна не позже чем через 300 единиц времени после своей активизации выдать управляющее воздействие на эффектор `force`. Поскольку других претендентов на процессорное время программное приложение не содержит, интерференции заданий не возникает, управляющее воздействие гарантированно выдается на эффектор `force` не позже чем через 150 единиц времени.

4.2. Многозадачные приложения. Реализация СРВ в виде примитивной однозадачной системы возможна в случае, если система либо содержит единственный активный датчик, либо вообще не содержит

активных датчиков и строится в виде последовательной программы с бесконечным циклом. Более типичны конфигурации внешнего оборудования с несколькими активными датчиками и, следовательно, с несколькими задачами типа обработчиков прерываний, выполняющих самые срочные функции СРВ. Менее срочные (и более сложные) функции возлагаются на программно-активизируемые задачи (потоки). Даже простейшую СРВ с составом внешнего оборудования, соответствующим спецификации в подразделе 2.3, целесообразно строить как двухзадачную: обработчик прерываний 1) считывает показания пассивных датчиков и 2) отправляет полученные показания в синхронизирующую очередь `input_data`. Вычисление и выдача управляющих воздействий выполняются программно-активизируемой прикладной задачей `regulator`.

```
<application>
    <isr name="rt_isr">
        <segment length="10"
                 interface="position"
                 op_type="get">
        </segment >
        <segment length="10"
                 interface="speed"
                 op_type="get">
        </segment >
        <segment length="10"
                 interface="input_data"
                 op_type="put">
        </segment >
    </isr>
    <thread name="regulator"
            prio="1" >
         <segment length="10"
                  interface="input_data"
                  op_type="get">
         </segment >
         <segment length="130"
                  interface="force"
                  op_type="put">
         </segment >
         <segment length="10"
                  interface="regulator">
         </segment >
    </thread
    < queue name="input_data"
```

```
            size="1"/>
</application>.
```

Так разделяются функции ввода информации от внешних объектов и функции выработки управления автоматизированным оборудованием. Первый сегмент задачи `regulator` соответствует ожиданию момента появления сообщения в очереди `input_data` и включает считывание сообщения. В рамках второго сегмента вычисляется и выдается на эффектор значение управляющего воздействия. Указатель эффектора `force` становится значением атрибута `interface` для второго сегмента потока. Для третьего сегмента значение `regulator` атрибута `interface` означает, что результатом выполнения сегмента является порождение нового экземпляра той же самой программно-активизируемой задачи.

Суммарная продолжительность интервалов времени от срабатывания активного датчика до выдачи управляющего воздействия не превышает 170 единиц времени. Поскольку допустимая продолжительность этого интервала равна 300 единицам, приложение выполнимо.

5. Приложения с мьютексами. Современный подход к организации взаимодействия программно-активизируемых заданий предполагает использование специальных методов доступа к критическим интервалам, охраняемым мьютексами. Для случая однопроцессорных СРВ разработаны точные методы оценки времени отклика задач в приложениях с фиксированными приоритетами задач и интерфейсными механизмами типа мьютексов. Выполнение этих методов требует учета фактора блокирования, зависящего от длин критических интервалов по доступу задач к разделяемым ресурсам. Представленные методы моделирования структуры кода задач средствами языка XML позволяют задавать длины критических интервалов, имеющихся в коде прикладной задачи. Пусть структура кода задачи представляется XML-моделью:

```
<thread name="task_0"  prio="0">
    <segment length="10"
             interface="mutex_0"
             op_type="get"/>
    <segment length="10"
             interface="mutex_1"
             op_type="get"/>
    <segment length="30"
             interface="mutex_0"
             op_type="put"/>
    <segment length="15"
```

```
                interface="mutex_1"
                op_type="put"/>
</thread>
```

Анализ структуры задачи показывает, что код задачи содержит пять сегментов. Первый сегмент заканчивается `get`-операцией над мьютексом `mutex_0`. Критический интервал по доступу к соответствующему ресурсу g_0 занимает участок кода от конца первого сегмента до конца третьего сегмента, завершающегося `put`-операцией над мьютексом `mutex_0`. Длина $C_0(g_0)$ этого критического интервала равна сумме длин второго и третьего сегментов. Критический участок по ресурсу g_1, контролируемому мьютексом `mutex_1`, состоит из третьего и четвертого сегментов, следовательно, $C_0(g_1) = 45$.

Алгоритмы проверки динамической корректности приложений СРВ по данным, представляемым рассмотренными XML-моделями, приведены, в частности, в работе [6].

6. Заключение. Изложенный в настоящей работе подход к спецификации интерфейсов в программных приложениях для систем реального времени позволяет строить формальные модели связей задач, составляющих программное приложение, между собой и с внешним оборудованием. Представление моделей интерфейсов средствами языка XML эффективно в отношении визуального восприятия специалистами, ведущими разработку, модификацию, сопровождение СРВ, а также в отношении автоматической обработки.

Из широкого арсенала изобразительных средств языка XML в рамках представленного подхода XML-код, специфицирующий интерфейсы программного приложения, использует только два типа конструкций — *содержание элемента* и *атрибут элемента*. Конструкция «атрибут элемента» используется для указания значений параметров моделируемых объектов. Конструкция «содержание элемента» используется для указания:

— множества датчиков и эффекторов, реализующих связь программного приложения с внешним оборудованием,

— множества задач, составляющих программное приложение,

— множества интерфейсных элементов, обеспечивающих межзадачные связи,

— последовательности сегментов кода в каждой из прикладных задач.

Такой ограниченный набор разновидностей используемых конструкций позволяет обрабатывать тексты спецификаций интерфейсов исключительно компактными инструментальными программами, компилирующими XML-описания в формы, ориентированные на эффективную реализацию алгоритмов анализа свойств моделируемых программных приложений.

Литература

1. *Котов В.Е.* Сети Петри. М.: Наука. 1984. 215 с.
2. *Буч Г., Якобсон А., Рамбо Дж.*. UML. СПб.: Питер, 2006. 736 с.
3. *Питц-Моултис Н., Кирк Ч.* XML. СПб.: БХВ-Петербург, 2001. 736 с.
4. *Давиденко К.Я.* Технология программирования АСУТП. Проектирование систем реального времени, параллельных и распределенных приложений. М.: Энергоатомиздат, 1985. 183 с.
5. *Данилов М.В.*. Методы планирования выполнения задач в системах реального времени // Программные продукты и системы. 2001. № 4. С. 28–35.
6. *Никифоров В.В., Гуцалов Н.В.* Анализ выполнимости составных задач в системах реального времени // Тр. СПИИРАН. 2005. Вып 2, т. 2. С. 437–452.

Никифоров Виктор Викентьевич — д-р техн. наук, проф.; ведущий научный сотрудник лаборатории технологий и систем программирования Санкт-Петербургского института информатики и автоматизации РАН. Область научных интересов: программирование систем реального времени, встроенных систем, операционные системы. Число научных публикаций — 80. nik@iias.spb.su; СПИИРАН, 14-я линия В.О., д.39, Санкт-Петербург, 199178, РФ; р.т. +7(848)328-0887.

Nikiforov Victor — Dr.Sc. (Tech.), Prof.; Senior researcher, Laboratory for Software Technology and Systems, St. Petersburg Institute for Informatics and Automation of the Russian Academy of Sciences (SPIIRAS). Research interests: methods for real-time software development, embedded systems, operating systems. The number of publications — 80. nik@iias.spb.su; SPIIRAS, 39, 14th Line V.O., St. Petersburg, 199178, Russia; office phone +7(848)328-0887.

Шкиртиль Вячеслав Иванович — канд. техн. наук, доц.; заведующий лабораторией технологий и систем программирования Санкт-Петербургского института информатики и автоматизации РАН. Область научных интересов: Программное обеспечение систем реального времени. Число научных публикаций — 60. jvatlas@mail.rcom.ru; СПИИРАН, 14-я линия В.О., д.39, Санкт-Петербург, 199178, РФ; р.т. +7(848)328-0887.

Shkirtil Viacheslav — Ph.D. (Tech.), associate professor; Dr.Sci., Head of the Laboratory for Software Technology and Systems, St. Petersburg Institute for Informatics and Automation of the Russian Academy of Sciences (SPIIRAS). Research interests: software development for real-time systems. The number of publications — 60. jvatlas@mail.rcom.ru SPIIRAS, 39, 14th Line V.O., St. Petersburg, 199178, Russia; office phone +7(848)328-0887.

РЕФЕРАТ

Никифоров В.В., Шкиртиль В.И. **Спецификация средствами языка XML системы интерфейсов в приложениях реального времени.**

Построение рациональной и корректной системы интерфейсов программных приложений относится к ряду важных конструктивных решений при разработке архитектуры систем реального времени (СРВ). На протяжении десятилетий развития технологии программирования разработчикам предлагались различные средства спецификации интерфейсов в программных системах — от всевозможных модификаций сетей Петри до UML-диаграмм. Предлагавшиеся средства в основном ориентированы на проверку логической корректности программных систем путем визуального, а также в той или иной мере автоматизированного анализа.

Для СРВ необходимо обеспечивать не только логическую, но и динамическую корректность программного комплекса, обеспечивать выполнение функций системы в заданных временных рамках. Проверка гарантий динамической корректности моделей программных приложений СРВ выполняется в два этапа: 1) модели, ориентированные на визуальное восприятие, представляются в аналитическом виде; 2) для автоматической обработки переводятся коды универсального языка программирования. Авторами разработан подход к представлению спецификаций системы интерфейсов средствами расширяемого языка разметки XML. Разработанный подход позволяет объединить эти два этапа. Средства XML могут быть использованы для построения таких формализованных моделей интерфейсов, которые в равной мере пригодны как для восприятия специалистами, выполняющими разработку, модификацию, сопровождение СРВ, так и для обработки автоматическими анализаторами, обеспечивающими, в частности, анализ выполнимости приложений, адекватных представляемым моделям.

Из широкого арсенала изобразительных средств языка XML в рамках представленного подхода XML-код, специфицирующий интерфейсы программного приложения, использует только два типа конструкций — *содержание элемента* и *атрибут элемента*. Конструкция «атрибут элемента» используется для указания значений параметров моделируемых объектов. Спецификации позволяют представлять множество задач, составляющих программное приложение; последовательность сегментов кода в каждой из задач; множество интерфейсных элементов, обеспечивающих межзадачные связи; множество датчиков и эффекторов, реализующих связь программного приложения с внешним оборудованием.

Такой ограниченный набор разновидностей используемых конструкций позволяет обрабатывать тексты спецификаций интерфейсов исключительно компактными инструментальными программами, компилирующими XML-описания в формы, ориентированные на эффективную реализацию алгоритмов анализа свойств моделируемых программных приложений.

SUMMARY

Nikiforov V.V., Shkirtil V.I. **Specification of interfaces in real-time software applications by XML forms.**

Creation of rational and correct system of software interfaces belongs to a set of inportant design decisions for real-time system (RTS) architecture development. Many means for software interface specification of had been sugested to system developers during the history of software design technology — from variouse modifications of Petry nets to UML-diagramms. Such means are oriented basically for checking of logical correctness of software systems by the way of visual analysis versus of analysis, more or less supported by computer.

In the case of RTS not only logical, but also dynamical correctness of software application should be guaranteed. The system functions should be implemented in predifined time restrictions. The checking of dynamic correctness of program application models is carried out by two stages: the models suitable for visual apprehending are built in analitical forms; for automatic processing they are transformed into the code of universal programming language.

Authors have developed an approach for use of XML means due to build specifications for interfaces in real-time applications. The developed approach permits to unite both stages. Formal models of interfaces that will built by the means of XML, are equally suitable either for visual apprehending by specialists or for automatic processing; this automatic processing may perform, particulary, the feasibility analysis of real-time applications, which are adequate to analised XML models.

From the wide set of XML form types only two are used in the frame of presented approach: XML specification of application interfaces contains only entities of *element contents* or *element attribute*. The forms «element attribute» present a set of object parameters. Components «element contents» present a set of tasks makes up software application; a set of interface elements that implement intertask links; a sequence of code segments in each task; a set of sensors and effectors that link software application with external environment.

Such restricted set of component types permits processing of interface specification by comcact instumental tools that compile XML description into machine-oriented form, that accomodate to automatic analysis of real-time software application features.