УДК 004.451.24,004.451.62

В. И. Межуев, канд. пед. наук
Бердянский государственный педагогический университет,
г. Бердянск

ИСПОЛЬЗОВАНИЕ МЕТАМОДЕЛИ ДЛЯ ПОСТРОЕНИЯ МОДЕЛЕЙ ВЗАИМОДЕЙСТВИЙ ЗАДАЧ В ОПЕРАЦИОННОЙ СИСТЕМЕ РЕАЛЬНОГО ВРЕМЕНИ

В статье строятся и исследуются свойства моделей взаимодействий задач в операционной системе реального времени OpenComRTOS. Главная особенность предложенного подхода состоит в использовании метамодели для построения моделей взаимодействий, обладающих заданными свойствами. В частности, рассмотрено возникновение эффекта синхронизации в случае, когда действия задач имеют различную временную семантику.

Ключевые слова: *модель, метамодель, взаимодействие задач, операционная система реального времени.*

Введение. OpenComRTOS (*Open Communication Real Time Operation System*) является операционной системой реального масштаба времени, предназначенной для использования во встроенных (*embedded*) системах [1, 2]. Основными понятиями OpenComRTOS являются ***задача*** (*task*) и характерные для параллельного программирования ***объекты синхронизации*** — порт (*port*), событие (*event*), семафор (*semaphore*), ресурс (*resource*), *FIFO* и др. Данные объекты синхронизации обобщаются в OpenComRTOS в едином понятии концентратора (*hub*).

В статье [3] нами была рассмотрена возможность порождения моделей распределенных параллельных приложений реального времени при помощи метамодели, построенной на основании теории графов. Узлами графа в этом случае являются задачи и объекты синхронизации OpenComRTOS, а ребрами — взаимодействия между ними. Такой подход позволяет использовать методы теории графов для решения задач рассматриваемой предметной области (ПрО), в частности, для генерации кода моделируемой программной системы.

Целью данной работы является расширение семантики моделей программных систем, порождаемых из основанной на теории графов метамодели. Для этого конкретизируется структура, а также вводятся дополнительные свойства узлов и ребер графа модели программного приложения. Такой подход позволяет породить семантики взаимодействий задач в распределенной параллельной системе, обладающие требуемыми свойствами.

В частности, выделим следующие необходимые свойства семантик взаимодействия задач:

- безопасность взаимодействия задач (гарантированное осуществление синхронизации);
- защита внутреннего состояния задач при их взаимодействии;
- возможность асинхронной работы в системе, обладающей ограниченными ресурсами;
- поддержка различной временной семантики взаимодействий;
- возможность построения объектов синхронизации, специфичных для предметной области;
- композиция более сложных программных систем из базовых элементов (задач и объектов синхронизации);
- возможность формальной проверки моделей программных систем.

Данная статья посвящена построению моделей взаимодействий задач в распределенной параллельной системе, обладающих заданными свойствами.

1. Основные идеи и структура работы

Рассмотрим ключевые принципы, которые позволяют построить модели взаимодействий задач, обладающие перечисленными выше свойствами. Прежде всего, это наложение ограничений на субъекты и объекты взаимодействий. Задачи OpenComRTOS являются *активными объектами*, которые могут взаимодействовать только через *пассивные объекты синхронизации* посредством пересылки особых структур данных — пакетов (*packet*). Преимущество данного подхода состоит в том, что объекты синхронизации разделяют взаимодействующие задачи. Под *разделением* мы понимаем семантику поведения, в которой задача не знает о другой задаче, с которой она взаимодействует. В процессе взаимодействия в пакете передается *копия* значений переменных задачи, что является *способом защиты внутреннего состояния задачи*.

Такой подход позволяет определить задачу OpenComRTOS как компонент. Выполняя *композицию* компонентов, осуществляющих взаимодействие через объекты синхронизации, можно создавать сложные параллельные системы. Для использования компонента достаточно знать лишь его *интерфейсы*; доступ к его внутреннему состоянию не является необходимым.

Этот способ построения параллельных систем был впервые формализован в алгебре взаимодействующих последовательных процессов Хоара — CSP (Communicating Sequential Processes) [4]. В CSP параллельная система состоит из *процессов* и *каналов*. Процесс выполняется при помощи (возможно бесконечного числа) последовательных шагов. Последовательность индивидуальных шагов (назы-

ваемых также следами) процессов разделяется коммуникацией через каналы. В случае, когда процессы синхронизированы на канале, через него могут быть переданы данные, что является простейшей формой взаимодействия. CSP подход является очень мощным и обеспечивает механизм для формального построения и проверки больших параллельных программных систем. Однако механизм синхронизации процессов через CSP канал весьма ограничен в своей семантике. Например, CSP каналы не учитывают параметры времени, хотя позже это было исправлено в так называемом Timed CSP [5].

Концентратор (*hub*) OpenComRTOS обобщает функциональные возможности канала CSP. Так же, как и в CSP, основной функциональной возможностью экземпляров *hub* (т.е. *port*, *event*, *semaphore*, *FIFO* и др.) является осуществление синхронизации задач. Однако *hub* осуществляет взаимодействие задач, основываясь на *результате проверки условия синхронизации*, что позволяет реализовать намного более сложную семантику взаимодействий.

Hub также может быть промежуточным звеном для взаимодействия большего, чем два, числа задач, тогда как канал CSP предназначен для синхронизации только двух процессов. Однако заметим, что поведение, аналогичное *hub*, может быть достигнуто в CSP путем введения специального промежуточного процесса между задачами.

Таким образом, объекты синхронизации OpenComRTOS — событие (*event*), семафор (*semaphore*), ресурс (*resource*), FIFO и т.д., определяют надмножество семантики CSP. Важное преимущество предложенного подхода состоит в том, что объекты синхронизации OpenComRTOS могут быть специализированными, т.е. определяемыми пользователем. Это позволяет порождать семантики взаимодействий задач, обладающие заданными свойствами.

В частности, отметим возможность реализации семантик ожидания, не ожидания, ожидания период времени, а также полностью асинхронных взаимодействий задач. Такой подход позволяет прикладному программисту выражать свойства ПрО способом, который наиболее соответствует ее (требуемому) поведению. Специфичные для ПрО объекты синхронизации могут быть порождены из *hub* путем указания условия синхронизации и действия, которое происходит в случае его истинности. В этом и состоит основное отличие нашего от традиционных подходов, которые требует полной перестройки ядра операционной системы для поддержки специфики ПрО.

Т.о. *hub* состоит из логического суждения (условия синхронизации) и действия синхронизации. Действие синхронизации осуществляется в случае истинности условия синхронизации. Проверка условия перед выполнением действия делает также возможным формальное

моделирование и проверку параллельных программных систем. Интерес представляет также декомпозиция условия синхронизации на прei постусловия. В этом случаем мы проводим аналогию между объектами синхронизации OpenComRTOS и тройками Хоара {P}C{Q} [6].

Раздел 2 данной статьи посвящен расширению подхода CSP путем введения модели взаимодействия задач через промежуточные объекты синхронизации. Раздел 3 исследует возможные временные семантики предложенной модели взаимодействия задач. Раздел 4 посвящен расширению семантики взаимодействий задач путем анализа связи модели со спецификациями языка TLA (Temporal Logic of Actions) [7] и тройками Хоара [6]. Завершают статью выводы и список использованных источников.

2. Моделирование семантики взаимодействия задач

Декомпозиция любой ПрО на объекты и их взаимодействия является естественным путем человеческого познания. Соответственно, все существующие методики моделирования ПрО подчеркивают использование объектов и их отношений. Естественным математическим средством для построения моделей подобного рода ПрО является теория графов.

Разрабатываемый нами подход находится в рамках идей предметно-ориентированного моделирования — DSM (Domain Specific Modelling). В этом контексте теория графов является основанием для метамодели, определяющей грамматику для построения моделей ПрО. Иными словами, теория графов предоставляет совокупность понятий (модельных объектов) и правил их объединения для построения моделей ПрО, обладающих требуемыми свойствами.

Как мы уже отмечали выше, задачи и взаимодействия OpenComRTOS типизируются нами соответственно как узлы и ребра графа. Построение обладающих требуемыми свойствами моделей требует более детального рассмотрения понятия *взаимодействия* задач.

Под *взаимодействием* мы понимаем, что *задачи осуществляют действия взаимным способом*, что часто считается побочным эффектом их коммуникации. Однако заметим, что взаимность действий не означает, что задачи выполняют их в сотрудничестве к достижению некоторой предопределенной цели.

Взаимодействие имеет место, когда два или более объектов действуют друг на друга. Таким образом, основная идея взаимодействия состоит в двухстороннем эффекте, в противоположность одностороннему причинному эффекту. Рис. 1 отражает такую схему взаимодействия двух задач (именованных здесь Задача$_1$ и Задача$_2$).

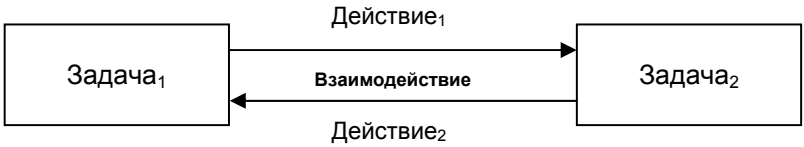

Рис. 1. *Схема взаимодействия задач*

Из этой простой графической модели следует, что взаимодействие задач должно включать по крайней мере два взаимно противоположных действия, именованных здесь Действие$_1$ и Действие$_2$.

Для гарантирования безопасности и реализации других требуемых принципов семантики взаимодействий задач мы помещаем между задачами объект синхронизации (обобщением которого является *hub*).

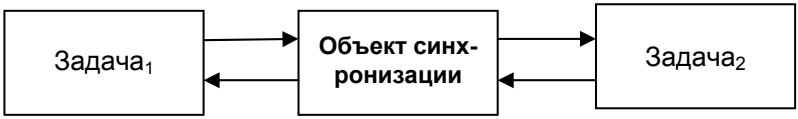

Рис. 2. *Взаимодействие через промежуточный объект синхронизации*

В этой модели взаимодействие между задачами состоит из двух симметричных действий между каждой задачей и объектом синхронизации. Эти действия будем называть посылкой (*put*) и получением (*get*), что является отражением имен сервисов, используемых в реализации для обработки пакетов.

Таким образом, процесс взаимодействия задач декомпозируется нами в пару *Put* (в дальнейшем для краткости будем использовать *P*) и *Get* (в дальнейшем будем использовать *G*) действий, приводящих к эффекту *S*, то есть синхронизации между задачами.

Сущность следующего утверждения состоит в том, что взаимные *P* и *G* действия двух задач есть необходимое условие их синхронизации (мы не говорим здесь ничего относительно временных свойств действий, порядка их следования и условий, определяющих каждое действие):

$$\left(P \wedge G\right) \vee \left(G \wedge P\right) \rightarrow S, \qquad (1)$$

где логическое «И» символизирует, что для синхронизации S должны иметь место оба действия.

Мы используем здесь понятие действия в толковании, аналогично данному Лесли Лампортом для TLA [7, с. 16]: «действие — обычная математическая формула …, действие является истинным или ложным на шаге».

Уравнение (1) можно также рассмотреть как следствие одного из принципов дизайна OpenComRTOS — симметрии механизма син-

хронизации, для реализации которого между взаимодействующими задачами помещается объект синхронизации.

Заметим, что задачи в параллельной системе имеют равные права для взаимодействия. Таким образом, по умолчанию, нет никакой главной задачи, которая служит причиной действий других задач. Задача, посылая пакет в объект синхронизации, не знает ничего о другой задаче, также взаимодействующей с данным объектом синхронизации. Иными словами, между взаимодействующими задачами не существует причинно-следственной связи.

Уравнение (1) определяет свойство, что только взаимные действия двух задач приводят к их синхронизации, и именно поэтому мы классифицируем этот процесс как взаимодействие.

Важный случай имеет место, когда осуществляется только одно действие — *Put* или *Get*. Если внешний наблюдатель обнаруживает только одно действие, то процесс синхронизации начинается, но не заканчивается. При использовании ждущего сервиса, задача, начавшая действие, перестает выполняться (или же, другими словами, блокируется).

Уравнения (2) и (3) отражают сценарий, когда задача$_1$ выполнила ждущее действие P_1, но задача$_2$ еще не осуществила действие G_2. Это приводит к тому, что задача$_1$ блокируется. Точно так же задача$_2$ блокируется после выполнения ждущего действия G_2, когда задача$_1$ еще не осуществила P_1.

$$P_1 \wedge G_2 \rightarrow Task_1\ blocked, \tag{2}$$

$$P_1 \wedge G_2 \rightarrow Task_2\ blocked. \tag{3}$$

Заметим, что уравнения (2) и (3) также отражают семантику взаимодействия процессов на ждущем (блокирующем) CSP канале.

Блокированная задача снова становится активной, если происходит *соответствующее* действие. Соответствующее означает, что тип действий задач должен быть противоположным: если первое действие имеет *Put* тип, то второе действие должно иметь *Get* тип. Синхронизация есть единственный способ продолжить выполнение блокированных задач.

3. Временные свойства взаимодействий задач

Уравнение (1) означает, что только взаимные действия (*взаимодействие*) задач приводят к синхронизации. В этом разделе мы обсуждаем значение понятия *взаимодействие* в контексте времени.

OpenComRTOS позволяет осуществить следующие временные семантики действий задач: ожидание (W), не ожидание (NW) и ожидание период времени (WT). Рассмотрим условия возникновения эф-

факта синхронизации в случае, когда действия задач имеют различную временную семантику.

Вообще говоря, временная семантика взаимодействий задач зависит от типа объекта синхронизации и граничных условий (например, граничного размера списка элементов FIFO). В этом разделе статьи будем использовать понятие синхронизации в CSP смысле, то есть как синхронную передачу данных по каналу связи между двумя задачами. Самым близким к функциональности канала CSP в OpenComRTOS является порт (*port*).

Исходя из трех вариантов действий задач, OpenComRTOS имеет 9 возможных случаев временной семантики взаимодействий двух задач.

Таблица 1

Варианты временной семантики взаимодействий задач в OpenComRTOS

Временная семантика	W (ожидание)	WT (ожидание период времени)	NW (не ожидание)
W (ожидание)	Симметрия	<W, WT>	<W, NW>
WT (ожидание период времени)	<WT, W>	симметрия	<WT, NW>
NW (не ожидание)	<NW, W>	<NW, WT>	симметрия

В случае, когда взаимодействия принадлежат к первому квадранту Таблицы 1 (т.е. оба действия используют ждущую семантику W), синхронизация не зависит от времени.

Случаи симметрии, представленные как диагональ Таблицы 1, имеют место, когда обе задачи используют один и тот же тип сервисов (то есть NW и NW, WT и WT, W и W). Свойство такой симметрии взаимодействий во времени состоит в том, что синхронизация задач не зависит от относительного порядка их действий.

В случае возникновения 1) <W, WT>, 2) <W, NW>, 3) <WT, NW> последовательностей синхронизация зависит от порядка действий задач. Для выражения временной семантики механизма синхронизации задач в таком случае должны использоваться операторы темпоральной логики (например, [7]).

1) Если W-действие произошло ранее WT-действия, синхронизация происходит в любом случае.
   Если сначала произошло WT-действие, то для синхронизации W-действие должно быть осуществлено во временном интервале T.
2) Если W-действие произошло прежде NW-действия, синхронизация происходит в любом случае.

Если NW-действие произошло до осуществления W-действия, синхронизация не произойдет.

3) Если WT-действие произошло до NW-действия, то для синхронизации NW-действие должно быть осуществлено во временном интервале T.

   Если NW-действие произошло до осуществления WT-действия, синхронизация не произойдет.

Заметим, что в формулировке всех этих утверждений, мы игнорируем способность объекта синхронизации к буферизации.

Такие элементарные последовательности действий посредством композиции могут быть применены для построения и анализа временных семантик сложных взаимодействий. Для этого элементарные последовательности взаимодействий должны быть сформулированы как временные формулы в синтаксисе TLA. Следующее состояние программной системы будет зависеть от результата предыдущих взаимодействий (приводящий к синхронизации или же блокирующий приложение).

Формализуем и рассмотрим свойства симметричных случаев синхронизации задач в OpenComRTOS.

NW-взаимодействие задач имеет следующую семантику:

$$\left(P\big|_{t=t1} \wedge G\big|_{t=t2}\right) \wedge \left(t_1 = t_2\right) \rightarrow S. \qquad (4)$$

Уравнение (4) определяет, что, в случае использования NW-сервисов, синхронизация S имеет место, если обе задачи осуществляют соответствующие действия P и G в один и тот же момент времени.

Поэтому, строго говоря, при возникновении <NW, NW> последовательности действий задач их синхронизация не является возможной, что является следствием их последовательного выполнения на машине архитектуры Фон Неймана. Иными словами, вычислительное устройство, на котором расположен объект синхронизации, преобразовывает доступ к нему в строго последовательную форму. Именно поэтому обработка двух NW-пакетов в одном и том же объекте синхронизации в один и тот же момент времени является невозможной, что и приводит к невозможности NW-синхронизации в *hub*.

Уравнение (4) может быть истинным только тогда, когда одно из действий (типа *Put* или *Get*) было буферизировано в списке ожидания *hub*. Например, используя NW-сервис можно получить пакет из FIFO, если он уже был помещен в него при помощи NW-сервиса ранее. (Заметим, что задача также может получить из FIFO собственный пакет, который был помещен ею в FIFO прежде.) Таким образом, NW-семантика синхронизации задач возможна в случае осуществления *hub* буферизации пакетов. Это является следствием того факта, что синхронизация задач является вторичным эффектом от успешно-

го завершения действий между *hub* и задачами, однако не непосредственно между задачами.

Вообще говоря, единственная безопасная семантика осуществления взаимодействий задач является блокирующей (ждущей) семантикой. Именно поэтому использование семантики ожидания — обычный путь для осуществления синхронизации задач, как это и практикуется в CSP. Использование NW-сервисов можно рассмотреть как способ проверки, является ли в данный момент синхронизация возможной (или же способом синхронизации, когда *hub* имеет поддержку буферизации).

WT синхронизация задач имеет следующую семантику:

$$\left(P\big|_{t=t1} \wedge G\big|_{t=t2}\right) \wedge \left(\left|t_2 - t_1\right| < T\right) \to S, \tag{5}$$

где T является интервалом времени.

Уравнение (5) определяет, что в случае использования WT-сервисов, синхронизация задач имеет место, когда промежуток времени между действиями P и G менее, чем интервал T.

Заметим, что формула (5) переходит в (4) в случае $T = 0$. В случае $T = \infty$ мы переходим к ждущей (W) семантике.

Приводящие к синхронизации S, W -действия задач имеют следующую семантику:

$$\left(P\big|_{t=t1} \wedge G\big|_{t=t2}\right) \wedge \left(\left|t_2 - t_1\right| < \infty\right) \to S \tag{6}.$$

Т.о. в случае использования W сервисов, промежуток времени, в течение которого может быть осуществлена синхронизация задач, является бесконечным. Уравнение (6) сводится к уравнению (1), то есть синхронизация происходит в случае взаимных действий задач вне зависимости от времени. Заметим, что этот тип аналогичен синхронизации на канале CSP.

Асинхронная семантика взаимодействий в OpenComRTOS также возможна, хотя такие взаимодействия в действительности являются *отложенными* синхронными взаимодействиями. Семантика асинхронных во времени действий задач может быть определена как «выстрелил и забыл». Заметим, что асинхронные взаимодействия требуют неограниченного числа ресурсов, однако ресурсы реальной системы всегда ограничены. Это налагает серьезные ограничения на возможность применений асинхронных взаимодействий.

Функциональные возможности *hub* обеспечивают иной способ осуществления асинхронного поведения задач. Например, для FIFO семантика ожидания будет иметь место, только если FIFO полон (достигнут предел буфера FIFO — *Size*), и мы имеем задачу посылки

(*Put*); или же если FIFO пуст (счетчик буфера FIFO равен нулю), и мы имеем задачу получения (*Get*).

$$\left(P \wedge Count\left(FIFO\right) = Size\right) \vee \left(G \wedge Count\left(FIFO\right) = 0\right) \rightarrow Wait \quad (7).$$

Следовательно, нормальное поведение осуществляющих буферизацию сервисов всегда является асинхронным, и сводится к синхронному только тогда, когда буфер полон или пуст. Осуществление такой семантики поведения задач в реальной системе является предпочтительным, так как она осуществляется при ограниченных ресурсах.

Таким образом, для FIFO для всех временных семантик действий (W, NW, WT) синхронизация имеет место независимо от времени, если формула (7) ложна.

Подобные свойства могут быть сформулированы для функциональных возможностей таких сущностей синхронизации как событие (*event*) и семафор (*semaphore*), которые также осуществляют асинхронное поведение.

Заметим, что в случае синхронизации задачи на *событии* или на *семафоре* важен порядок осуществления действий. Синхронизация на *событии* имеет место, если первым во времени действием является *Put* (применительно к *событию*, сервис именуется *RaiseEvent*), с последующим *Get* (*TestEvent*), с условием, что изначально событие не было взведено. Обе задачи могут продолжить выполнение немедленно после осуществления их синхронизации через событие, независимо от временной семантики действий.

Описывая семантику механизма синхронизации задач, мы неявно используем предположение, что она может иметь место только на том же самом объекте синхронизации:

$$Hub\left(P\right) = Hub\left(G\right). \quad (8)$$

Уравнение (8) отражает пространственную семантику механизма синхронизации задач в распределенной параллельной системе.

4. Модель объекта синхронизации

Как было отмечено нами выше, пользователь может определить собственный объект синхронизации в рамках его модели, получившей название концентратора (*hub*).

В отвлечении от технических деталей, *hub* включает предикат синхронизации и действие синхронизации.

$$Hub \overset{def}{=} Predicate \wedge Action. \quad (9)$$

Исходя из определения (9) модель параллельной программы в OpenComRTOS может быть определена как машина состояний, выполняющая действия только тогда, когда условия синхронизации задач яв-

ляются истинными. Необходимыми слагаемыми условий синхронизации есть временные свойства, рассмотренные нами в разделе 3 статьи.

Язык TLA [7] является основой формальной спецификации OpenComRTOS [1; 2]. Заметим, что определение (9) соответствует формулировке спецификации системы в TLA, которая также включает условия и действия [7].

$$A_0 = Guard_0 \wedge Action_0$$
$$\dots \tag{10}$$
$$A_{j-1} = Guard_{j-1} \wedge Action_{j-1}$$

Следующее состояние системы определяется как результат логической операции «или» между всеми возможными защищенными действиями задач.

$$Next = A_0 \vee A_1 \vee \dots \vee A_{j-1}. \tag{11}$$

Таким образом, мы можем определить протокол взаимодействия задач OpenComRTOS как последовательность защищенных действий, имеющих место во всех *hub* системы. Данный подход описывает поведение программной системы в терминах изменения ее внутреннего состояния. Состояние программной системы мы рассматриваем как объединение множеств состояний задач и объектов синхронизации. Последовательность действий $< A_1,\ A_2, \dots, A_n >$ задач, вызывает обновление состояний $< H_1,\ H_2, \dots, H_m >$ объектов синхронизации. Отвечающая (1) пара действий приводит к синхронизации задач и, таким образом, взаимодействие является механизмом, который вызывает переход состояний вовлеченных задач. Только синхронизация позволяет задачам выполняться далее, поэтому последовательность взаимодействий $< I_1,\ I_2, \dots, I_k >$ задач приводит к обновлению состояний $< P_1,\ P_2, \dots, P_l >$ программы.

Заметим, что первой логической системой, разработанной для проверки компьютерных программ, была логика Флойда и Хоара [6; 8]. Идея логики Хоара состоит в помещении предикатов в начало и конец блока программы. Аксиомы логики Хоара определяют входные предикаты как достаточные условия, гарантирующие, что успешное выполнение блока программы приводит к указанным выходным условиям.

Такие предикаты входа/выхода описывают поведение программы в альтернативной форме. Отношения между предикатами устанавливаются аксиомами языка программирования. Такая аксиоматическая проверка является методом доказательства правильности программы.

Например, предикаты для данных помещаются в различные точки программы и остаются инвариантами в процессе ее выполнения.

В этом подходе аксиоматическая семантика языка программирования определяется тройками Хоара (12). Операторы программы преобразуются в предикаты, а валидность графа программы сводится к валидности лемм, не содержащих операторы программы.

Поэтому следующим шагом в разработке модели *hub* является его рассмотрение в терминах троек Хоара [6]. Эти тройки выражаются следующим отношением:

$$\{P\}\, C\, \{Q\}, \tag{12}$$

где P является предусловием, Q — постусловием и C — командой языка программирования.

Это требует расширения модели *hub* путем декомпозиции предиката синхронизации в пред- и пост-условия. Предусловие синхронизации в *hub* разрешает действие, приводя к постусловию.

$$P\ recondition \wedge Action \rightarrow Postcondition \tag{13}$$

В качестве примера рассмотрим синхронизацию задач посредством семафора.

Предусловия:

- существует, по крайней мере, одна задача, ожидающая сигнала семафора;
- семафор установлен (т.е. его счетчик более единицы).

Действие (синхронизация):

- состояние ждущей задачи сделать активным;
- декрементировать счетчик семафора.

Постусловия:

- задача активна (выполняется снова);
- счетчик семафора уменьшен на единицу и его значение больше нуля.

Выводы. В работе предложены способы расширения семантики моделей программных систем, порождаемых из основанной на теории графов метамодели. Выделены необходимые свойства семантик взаимодействия задач в распределенной параллельной системе и предложен подход для построения моделей, которые обладают заданными свойствами. Основным средством для этого является введение модели объекта синхронизации (*hub*). Пользователь может определить собственный объект синхронизации путем указания условия синхронизации и действия, которое происходит в случае его истинности. Такой подход позволяет прикладному программисту выражать свойства ПрО способом, который наиболее соответствует ее (требуе-

мому) поведению. Построение *hub* на формальной основе связывает программирование с математическими методами и является шагом к проектированию формально доказанных программ.

Список использованной литературы:

1. Verhulst E. An Industrial Case: Pitfalls and Benefits of Applying Formal Methods to the Development of a Network-Centric RTOS / E. Verhulst, G. Jong, V. Mezhuyev // Lecture Notes in Computer Science. FM 2008 : Formal Methods. — Heidelberg : Springer Berlin, 2008. — P. 411—418.
2. Verhulst E. OpenComRTOS : A Runtime Environment for Interacting Entities / E. Verhulst, V. Mezhuyev, Bernhard H. C. Sputh [et al.]; P. Welch, H. Roebbers, T. Announced (eds.) // Communicating Process Architectures 2009. — IOS Press, 2009. — P. 173 — 184.
3. Межуев В. И. Предметно-ориентированное моделирование распределенных параллельных приложений реального времени / В. И. Межуев // Системи обробки інформації. — 2010. — Вип. 5 (86). — С. 98—103.
4. Hoare C. A. R. Communicating Sequential Processes. Published by Prentice Hall / C. A. R. Hoare // International Series in Computer Science. — 1985. — 276 p.
5. Timed CSP : A Retrospective / J. Ouaknine, S. Schneider // Electronic Notes in Theoretical Computer Science. — 2006. — 162 (1). — P. 273—276.
6. Hoare C. A. R. An Axiomatic Basis for Computer Programming / C. A. R. Hoare // Communications of the ACM. — Vol. 12, N 10. — P. 576—583.
7. Lamport L. Specifying systems : the TLA+ language and tools for hardware and software engineers / L. Lamport // Addison-Wesley, Boston. — 2002. — 364 p.
8. Floyd R. W. Assigning meanings to programs / R. W. Floyd, J. T. Schwartz (ed.) // Proc. of Symposium on Applied Mathematics. — 1967. — Vol. 19. — P. 19—32.
9. Rosenband D. Modular Scheduling of Guarded Atomic Actions / D. Rosenband // Proc. of the 41st Design Automation Conference (DAC'04) (San Diego, June 7—11, 2004). — San Diego, California, USA, 2004. — P. 8.

The models of tasks interactions in the OpenComRTOS real time operation system are developed and investigated in the paper. The main feature of the proposed approach is the use of a metamodel for development of models of the interactions, which having the needed properties. In particular, occurrence of a synchronization effect in the case when actions of tasks have different time semantics is considered.

Key words: *model, metamodel, tasks interaction, real time operation system.*

Отримано 22.05.10