УДК 004.413

Разработка процессов синхронизации моделей и принципов проверки их корректности

© В.В. Девятков, Д.В. Ошкало

МГТУ им. Н.Э. Баумана, Москва, 105005, Россия

Изложены новые принципы решения задачи синхронизации моделей на основе формального процессного описания механизмов синхронизации и свойств их корректности, формализуемых на языке временной модальной логики с последующей проверкой этих свойств с использованием логического вывода. Рассмотрены основные свойства, обеспечивающие корректность процесса синхронизации моделей, а также особенности создания систем синхронизации.

Ключевые слова: *синхронизация моделей, трансформация моделей, свойства корректности, процессы, модальная логика.*

Введение. Моделеориентированный подход к разработке программного обеспечения предполагает использование множества формальных структурированных компонентов — моделей — с целью автоматизированного построения на их основе различных артефактов — программного кода, документации, конфигурации и др. Используя подобный подход, разработчик избавляется от рутинной работы по написанию шаблонов и заготовок кода, созданию конфигурационных файлов и скриптов и т. п.

Однако при использовании моделеориентированного подхода на плечи разработчика ложится ряд других задач, связанных с созданием и сопровождением моделей, количество и разнообразие которых в зависимости от сложности проекта может быть значительным: модели требований, архитектуры компонентов, сценарии использования, модели данных и т. д. Зачастую эти модели так или иначе связаны друг с другом. Они могут разрабатываться отдельно, но при этом быть семантически идентичными или характеризовать с разных позиций один и тот же проект. Например, это могут быть различного рода концептуальные модели проекта [1] или модели, полученные на основе других моделей посредством трансформации, это может быть схема реляционной базы данных, созданная на основе диаграммы классов UML [2].

Механизмом синхронизации моделей, или просто синхронизацией, обычно называют механизм, имеющий дело с двумя моделями и осуществляющий внесение изменений в одну модель при изменениях в другой с сохранением корректности синхронизации. Понятие кор-

ректности синхронизации требует более строгого определения и будет дано позже.

В настоящее время наиболее распространенной является синхронизация, осуществляющая трансформацию одной модели в другую таким образом, что последняя уничтожается, а вместо нее создается новая версия. Такая синхронизация неработоспособна в тех случаях, когда изменения были внесены независимо в обе модели. Для этого случая требуется другой механизм синхронизации.

Кроме того, синхронизация может затрагивать более двух моделей, однако мы ограничимся механизмом синхронизации двух моделей, поскольку он является основой для других синхронизаций.

Универсальный (пригодный для всех случаев) механизм синхронизации пока не разработан. Рассмотрение же отдельных механизмов синхронизации является, на наш взгляд, излишним. Поэтому остановимся на том, что их объединяет.

Основные подходы к разработке механизмов синхронизации моделей можно разбить на две группы: декларативные и императивные (операционные).

Декларативные подходы основаны на использовании языков описания механизмов синхронизации, позволяющих представить механизм в виде множества правил таким образом, чтобы они гарантированно были безопасными с точки зрения определенных критериев. Для этого требуется точная формулировка критериев и процедуры проверки правил на соответствие им. Создание универсальных языков описания правил синхронизации и процедур проверки критериев правильности приблизило бы решение проблемы синхронизации моделей. Однако универсального языка синхронизации не существует. Наиболее удачным примером можно считать QVT-Relations, разработанный OMG и являющийся на сегодня стандартом описания трансформаций моделей[1, 3]. Средствами данного языка можно описывать правила трансформации и двунаправленного переноса изменений между моделями, но многие правила синхронизации в нем не реализованы или реализованы не полностью. Это обусловливает создание предметно- и проблемно-ориентированных языков для выполнения каких-либо специфических задач синхронизации [4].

Императивный подход к проектированию систем синхронизации предполагает создание алгебры преобразований, так как операции трансформации моделей по своей сути являются алгебраическими операциями над моделями и их элементами [5–10]. Операции и их свойства, как правило, зависят от конкретной предметной области и требований, предъявляемых к системе.

Известны попытки объединения декларативного и императивного подходов [7, 11].

Этому посвящена и наша работа. Для создания механизмов синхронизации предлагается использовать процессные модели [12], выразительные возможности которых широки, а теория хорошо развита, что позволяет описывать сложные механизмы синхронизации. Свойства механизмов синхронизации предлагается описывать формально как свойства процессных моделей синхронизации на языке временной модальной логики. Процессные модели в дальнейшем будем называть просто процессами. Процессы позволяют более четко и полно классифицировать и описывать синхронизацию. Анализ свойств синхронизации предлагается осуществлять методами логического программирования, т. е. путем доказательства или вывода свойств процессов синхронизации. Такой подход к проверке корректности синхронизации является гораздо более изящным, полным и не имеет ограничений.

Процессные модели. Аппарат теории процессов хорошо подходит для формального описания и анализа поведения систем, которые состоят из нескольких взаимодействующих компонентов. Эти компоненты работают параллельно, их взаимодействие происходит путем пересылки сигналов или сообщений от одних компонентов другим [13]. Такой системой является и система синхронизации моделей. Рассмотрим элементы аппарата теории процессов, которые будут применены в дальнейшем для описания ее поведения.

Каждый процесс имеет алфавит восприятий и реакций $A = \{a_1,\ a_2,\ ...,\ a_m\}$. Каждый символ a этого алфавита обозначает некоторый объект, получаемый (воспринимаемый) процессом из внешней среды (восприятие процесса), выдаваемый процессом во внешнюю среду (внешняя реакция процесса), или объект, используемый процессом для внутренних нужд (внутренняя реакция процесса). Процессы действуют, воспринимая, порождая для внутреннего употребления или выдавая наружу объекты с соответствующими именами. Для того чтобы различать типы действий, будем использовать следующие обозначения: *?a* — восприятия, *!a* — внешние реакции, *b* — внутренние реакции.

Нитью a^* будем называть кортеж (конечный или бесконечный) действий $a^* = a_0 a_1 a_2 ... a_{m-2} a_{m-1}$. Последовательность выполнения действий нити слева направо называется выполнением нити процессом *P*. Символом *e* обозначается пустое действие. Нить, состоящая из единственного пустого действия, называется пустой нитью. Процессом *P* называется множество нитей *S*, которые он может выполнять. Поведением процесса *P* называется порядок выполнения множества этих нитей.

Введем обозначения: *?A* — множество всех восприятий некоторого процесса *P*, включая пустое восприятие *?e*; *!A* — конечное мно-

жество всех внешних реакций процесса P, включая пустую внешнюю реакцию $!e$; S — множество всех нитей, выполняемых процессом P, таких что $S \subseteq ?A^* \times !A$, $?A^*$ — множество всех нитей $?a^*$ в алфавите $?A$. Будем обозначать $P(S)$ процесс P, выполняющий множество нитей $?a^*\varphi^*(?a^*) \in S$, где φ^* — функция на множестве $?A^*$, которая ставит в соответствие каждой нити $?a^* \in ?A^*$ внешнюю реакцию из множества $!A$. Значение функции φ^* на нитях множества $?A^*$, на которых эта функция не определена и которые этому множеству не принадлежат, считается равным $!e$. Множество нитей $?a^*$, таких что $\{?a^* \in ?A^* \mid ?a^*\varphi^*(?a^*) \in S\}$, будем обозначать $?S$.

Популярным языком представления процессов является язык графов переходов. При построении графа переходов процесса считается, что после каждого его восприятия, в том числе пустого, процесс осуществляет внутреннюю реакцию или, как говорят, переходит во внутреннее состояние ожидания следующего восприятия. Находясь в этом внутреннем состоянии, процесс может порождать внешнюю реакцию, в том числе пустую. В графе переходов каждое состояние процесса изображается в виде кружка, внутри которого — символ этого состояния; каждому восприятию соответствует стрелка, соединяющая состояния этого перехода. Начальное состояние выделяется двойным кружком. На рис. 1 показан граф переходов некоторого процесса P.

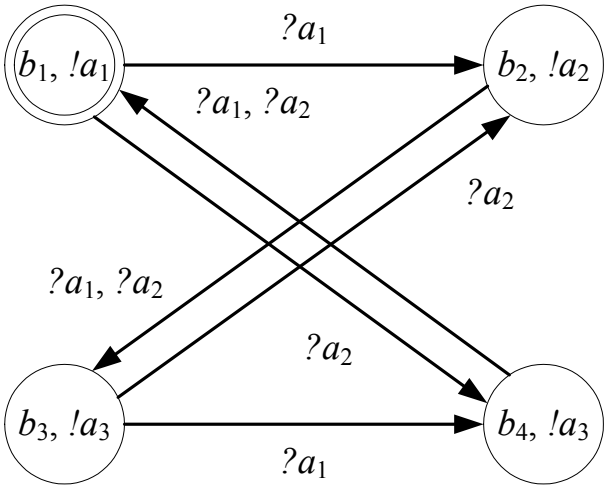

Рис. 1. Граф переходов процесса P

Граф переходов процесса позволяет компактно описывать небольшое множество нитей, в том числе бесконечных. В этих услови-

ях естественным кажется описывать процесс непосредственно его графом переходов. Но при большем размере графа такое описание становится громоздким и ненаглядным. Альтернативой являются канонические процессные выражения, позволяющие легко переходить от графа к этим выражениям и наоборот.

Так, для графа переходов на рис. 1 соответствующими каноническими процессными выражениями будут следующие:

$$P = \begin{pmatrix} b_1 = ?e \\ b_1 = b_4?a_1 \\ b_1 = b_4?a_2 \\ b_2 = b_1?a_1 \\ b_2 = b_3?a_2 \\ b_3 = b_2?a_1 \\ b_3 = b_2?a_2 \\ b_4 = b_1?a_2 \\ b_4 = b_3?a_1 \\ !a_1 = b_1 \\ !a_2 = b_2 \\ !a_3 = b_3 \mid b_4 \end{pmatrix}.$$

В этих выражениях каждое внутреннее процессное выражение вида $b_i = b_j?a_k$ задает один переход из состояния b_j в состояние b_i в результате восприятия $?a_k$. Выражение $!a_i = b_j$ задает реакцию $!a_i$ после перехода процесса в состояние b_j.

Очевидно, что если исходным описанием процесса являются канонические процессные выражения, то переход от этих выражений к графу переходов процесса и наоборот очевиден. Канонические процессные выражения могут быть рекурсивными.

При помощи обозначенного теоретического аппарата можно условно представить систему синхронизации моделей в виде процесса, воспринимающего пару моделей *?s* и *?t*, результатом работы которого является преобразованная определенным образом пара моделей *!s* и *!t* (рис. 2). Внутренние состояния *b* процесса синхронизации могут быть связаны с состоянием моделей или же с этапами их обработки.

Наиболее важный класс задач, для решения которых предназначена теория процессов, связан с проблемой верификации процессов. Она заключается в построении формального доказательства того, что анализируемый процесс обладает заданными свойствами.

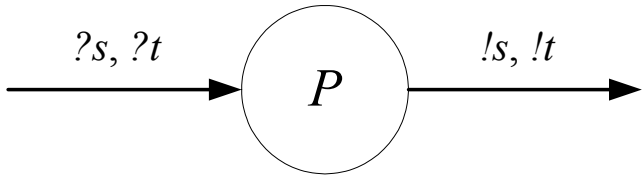

Рис. 2. Представление синхронизации моделей в виде процесса

Точная постановка задачи верификации состоит из следующих частей:

1. Построение процесса, представляющего собой математическую модель поведения анализируемой системы.

2. Представление проверяемого свойства в виде математического объекта (называемого спецификацией).

3. Построение математического доказательства утверждения о том, что построенный процесс удовлетворяет спецификации [13].

Свойства корректности процессных моделей синхронизации. Обозначим $(s,t) = R(s,t)$ отношение консистентности между моделями s, t [3]. Консистентность между моделями означает, что информация, представленная в одной из них, в определенном смысле не противоречит информации в другой. Отношение консистентности в каждом конкретном случае требует более детального определения.

Выделим свойства корректности процессных моделей синхронизации, записывая их на языке временной модальной логики [1].

Детерминированность. Детерминированность означает предсказуемость результата процесса синхронизации. Данное требование вытекает из соображений, что синхронизация моделей не изолирована от процесса проектирования и разработчик должен точно знать, к чему приведет результат очередной синхронизации. В частности, для одних и тех же моделей результат синхронизации должен быть одинаковым:

$$\{(?s, ?t) \supset \Diamond[((!s', !t') \vee (!s'', !t'')) \wedge (?s' \equiv !s'') \wedge (?t' \equiv !t'')]\}.$$

Детерминированность может быть сформулирована с различной степенью строгости. В работе [2] говорится о том, что детерминированность всегда должна предполагать единственный возможный вариант трансформации элементов между моделями. Принимать ли требование к детерминированности в таком виде, зависит от конкретного случая реализации системы синхронизации.

Стабильность. В дополнение к детерминированности приводят свойство стабильности [6], подразумевающее, что процесс синхронизации не должен изменять модель, если изменений в модели не было

сделано. Данное утверждение касается обеих синхронизируемых моделей:

$$\{(?s, ?t) \supset \Diamond[(!s, !t) \wedge (?t \equiv !t) \wedge (?s \equiv !s)]\}.$$

Симметричность. Данное свойство лежит в основе принципа равноправия моделей по отношению к процессу синхронизации: результат синхронизации не должен зависеть от порядка обработки моделей:

$$\{[(?s, ?t) \wedge (?s', ?t') \wedge (?s \equiv ?t') \wedge (?t \equiv ?s')] \supset \Diamond$$
$$[(!s, !t) \wedge (!s', !t') \wedge (!s \equiv !t') \wedge (!t \equiv !s')]\}.$$

Возможность отката изменений. Это требование проявляется в следующем сценарии. Исходная модель была модифицирована, и связанные с ней изменения были перенесены в целевую модель, таким образом обе модели стали консистентными. После этого изменения в первой модели были отменены, она вернулась к своему первоначальному состоянию. В этом случае целевая модель также должна вернуться к своему первоначальному состоянию:

$$\{[(?s, ?t) \wedge [(!s', !t') \wedge ((?s' \equiv ?s) \vee (?t' \equiv ?t))] \supset \Diamond$$
$$[(!s' \equiv ?s) \wedge (!t' \equiv ?t)]\}.$$

Принцип Гиппократа. Изменения, вносимые процессом синхронизации в структуру моделей, должны быть минимальными среди тех, что приведут модели к консистентному состоянию. Стабильность и возможность отката изменений являются частными формами данного принципа.

Принципы создания процессных программ синхронизации и проверки их корректности. Помимо обозначенных свойств корректности, которым должен удовлетворять процесс синхронизации моделей, можно выделить более частные свойства, обусловленные той или иной предметной областью, напрямую влияющие на внутреннюю логику процесса синхронизации. В работе [14] приведены возможные примеры систем синхронизации в зависимости от следующих критериев:

1. Гомогенность или гетерогенность синхронизируемых моделей. Гомогенными являются модели, состоящие из одних и тех же структурных элементов и построенные по одним и тем же правилам (т. е. однотипные модели). В противном случае модели являются гетерогенными (разнородными).

2. Однонаправленность или двунаправленность трансформаций. В зависимости от назначения может осуществляться как однонаправ-

ленная, так и двунаправленная синхронизация моделей. Реализация однонаправленных систем синхронизации более проста, так как не предполагает ни механизмов проверки целостности моделей, ни разрешения конфликтных ситуаций, которые при данном сценарии синхронизации возникнуть не могут.

3. Разрешение конфликтных ситуаций. Данный критерий обусловливает возможность нахождения и разрешения конфликтных ситуаций в процессе синхронизации. Он актуален только для синхронизации с двунаправленным переносом изменений.

4. Результаты синхронизации. Этот критерий накладывает существенный отпечаток на то, каким образом будет вести себя синхронизатор будет ли результат синхронизации зависеть только от текущего состояния моделей или будет учитываться история изменений[9].

5. Инкрементальность. Инкрементальность предполагает перенос изменений только тех компонентов моделей, которые были модифицированы. Компоненты, оставшиеся неизменными, в процессе синхронизации не участвуют. Возможность инкрементальной синхронизации особенно важна при синхронизации больших моделей, так как помогает значительно сэкономить время.

6. Выбор стратегии трансформации. Когда отношение консистентности между моделями является отношением типа «многие ко многим» (например, в случае синхронизации объектно ориентированного и реляционного форматов данных), необходим механизм определения того, как именно трансформировать элементы моделей. Возможно несколько вариантов реализации выбора стратегии: с участием пользователя, посредством жестко заданных условий, на основании информации о структуре моделей и т. д. Конкретных предложений по реализации подобного механизма в настоящий момент не существует, а любая информация, затрагивающая этот вопрос, носит описательный и рекомендательный характер.

Сегодня разработано более десятка разновидностей архитектур систем синхронизации моделей в зависимости от приведенных выше параметров, однако их детальная реализация — предмет дальнейших исследований.

Как правило, при проектировании систем синхронизации моделей сначала следует определить, какие модели эта система будет связывать и какими свойствами обладать. От этого зависит структура состояний процесса синхронизации и переходы между ними. Заметим, что создаваемая система необязательно должна удовлетворять всем рассмотренным свойствам корректности. В зависимости от предметной области и желаемого механизма синхронизации те или

иные свойства могут быть исключены как лишние или излишне строгие, а влияние некоторых может быть учтено в меньшей степени [6]. Далее требуется определить, какие задачи стоят перед создаваемым модулем синхронизации и какое качество процесса синхронизации должно быть в итоге достигнуто. В зависимости от этого выбирается тип архитектуры системы синхронизации, а также дополнительные компоненты, взятые в зависимости от параметров системы.

Примеры процессных программ синхронизации и проверки их корректности на языке ПРОЛОГ. Как было отмечено, важную роль в создании процессов синхронизации моделей играет их верификация. Используя полученные формальные выражения для свойств корректности синхронизации, а также определяя процессное представление данной процедуры, можно составить тестовую программу проверки корректности процесса синхронизации на одном из языков логического программирования.

Рассмотрим простейшую синхронизацию как совокупность двух не взаимодействующих друг с другом процессов, каждый из которых работает с одной из моделей, определенным образом изменяя ее состояние (рис. 3).

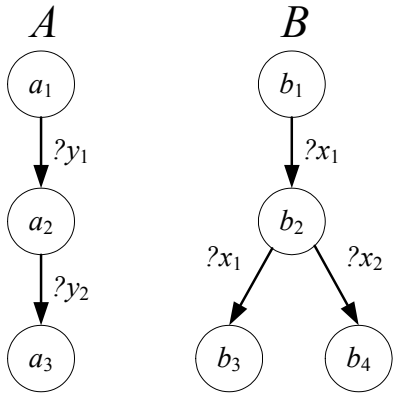

Рис. 3. Представление синхронизации моделей в виде двух параллельных процессов *A* и *B*

Представим эти процессы в виде логической программы (листинг 1) на языке ПРОЛОГ в версии SWI PROLOG [15]. Каждый переход процессов в этой программе представлен фактом, начинающимся с предикатного символа *transform*, отношение консистентности между моделями представлено перечислением пар состояний моделей, по достижении которых будем считать, что процесс синхронизации успешно завершен. Правила достижимости состояний каждого процесса представлены правилами, начинающимися с предикатов *accessible*.

Листинг 1

Пример процессной программы синхронизации на языке SWI PROLOG

```
% a1, a2, a3 — состояния процесса обработки модели A
% y1, y2 — воздействия, обусловливающие переход между
состояниями процесса обработки модели A
% b1, b2, b3, b4 - состояния процесса обработки модели B
% x1, x2 — воздействия, обусловливающие переход между
состояниями процесса обработки модели B

% правила переходов для процесса, работающего с моделью A
transform(a1, y1, a2).
transform(a2, y2, a3).
% правила переходов для процесса, работающего с моделью B
transform(b1, x1, b2).
transform(b2, x1, b3).
transform(b2, x2, b4).
% отношение консистентности
r(b4, a3).
r(b1, a3).
r(X,Y) :- r(Y,X).

% правила достижимости
accessible(B1, [H], B2) :- transform(B1, H, B2).
accessible(B1, [H|T], B2) :-  transform(B1, H, B3),
                              accessible(B3, T, B2).
% процесс синхронизации; XN, XY — начальные состояния
моделей A и B
sync(XN, YN, X,Y) :-   accessible(XN,[H1|T1], X),
                       accessible(YN,[H2|T2],Y), r(X,Y).
```

Используя правила достижимости, проверим, возможна ли корректная синхронизация моделей A и B с применением данных процессов. Для этих целей составим запрос

```
? — sync(b1, a1, b4, a3)
true
```

На запрос получен положительный ответ. Таким образом, для выбранных начальных и конечных состояний возможна синхронизация, в основе которой находится изображенный на рис. 3 процесс.

Далее рассмотрим пример алгоритма синхронизации, предложенного в [6]. Даны две модели A и B, каждая из которых имеет по два состояния: $\{a,b\} \in A$, $\{x,y\} \in B$. На некотором подмножестве со-

стояний моделей задано отношение консистентности. Начальными данными для процесса являются исходное и модифицированное в некоторый момент времени состояние каждой модели, причем в исходном состоянии модели являются консистентными. Процесс синхронизации представляет собой совокупность последовательных процессов трансформации модели *B* в модель *A* (*btrans*), обновления модели *A* (*merge*), а также трансформации обновленной модели *A* в модель *B* (*ftrans*) (рис. 4).

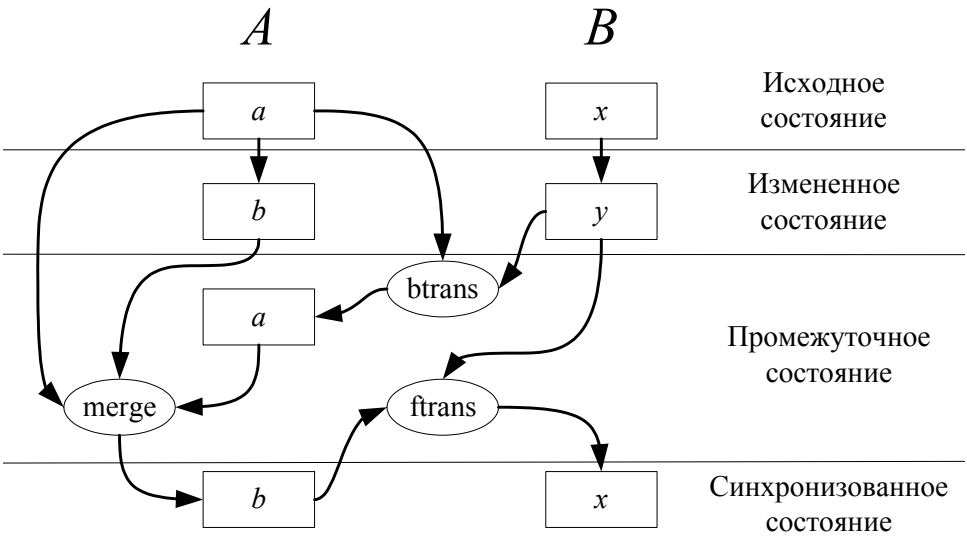

Рис. 4. Синхронизация моделей *A* и *B* путем двунаправленных трансформаций

В данном примере, в отличие от предыдущего, состояния процесса синхронизации ассоциированы с состояниями моделей, а не со стадиями процессов обработки моделей, поэтому выражения для смены состояния имеют другой вид (листинг 2).

Листинг 2

Синхронизация моделей, основанная на двунаправленных трансформациях моделей, реализованная на языке SWI PROLOG

```
% отношение консистентности моделей
r(a, x).
r(x, a).
r(a, y).
r(y, a).
r(b, x).
r(x, b).
r([X],[Y]) :- r(X,Y).
```

```
% описание процессов изменения моделей
% списки используются для сохранения информации о текущем
состоянии моделей
btrans(X,Y,List) :- r(Y,Z), append([Z],[],List).
merge(X,Y,Z,List) :- (X == Z -> append([Y],[],List) ;
                        append([Z],[],List) ).
ftrans(X,Y,List) :- r(X,Z), append([Z],[],List).

% процедура синхронизации
% X1, Y1 — состояние моделей до изменения
% X2, Y2 — состояние моделей после изменения
snc(X1, X2, Y1, Y2, List, List2) :- btrans(X1, Y2, [H]),
                    merge(H,X1,X2,[List]),
                    ftrans(List,Y2,[List2]),
                    r([List],[List2]).

% свойства процесса синхронизации
% симметричность
symmetric(X1,X2,Y1,Y2) :-  snc(X1,X2,Y1,Y2,List,List2),
                        writef("1:"),nl,
                        write(List),nl,write(List2),nl,
                        writef("2:"),nl,
                        snc(Y1,Y2,X1,X2,List3,List4),
                        write(List3),nl,write(List4),
                        List == List3,
                        List2 == List4.

% стабильность
stable(X1,X2,Y1,Y2) :- r(X1,Y1),
                        snc(X1,X2,Y1,Y2,List,List2),
                        writef("Результат:"),nl,
                        write(List),nl,write(List2),nl,
                        X1==List,
                        Y1==List2.
```

Проверим, удовлетворяет ли построенный таким образом процесс синхронизации требованию стабильности:

```
?— stable(a, a, x, x), stable(b, b, x, x), stable(a,a,y,y).
Результат:
      a
      x
```

```
Результат:
      b
      x
Результат:
      a
      y
true
```

Далее проверим свойство симметричности:

```
?- symmetric(a,b,x,y).
1:
y
a
2:
b
x
false
```

Итак, можно сделать вывод о том, что данный процесс синхронизации не обладает свойством симметричности: для него важен порядок выполнения операций над моделями.

Таким образом, рассмотрение механизмов синхронизации в терминах процессов и использование формальных описаний свойств процессных моделей с целью анализа процесса синхронизации в целом позволяют по заданному описанию процесса синхронизации построить систему проверки свойств его корректности, верификации и проанализировать его поведение в различных условиях.

Заключение. В статье поставлена и освещена проблема синхронизации моделей при моделеориентированной разработке программного обеспечения. Рассмотрены основные свойства процессов синхронизации, установлена их взаимосвязь и влияние на процесс проектирования систем синхронизации моделей. Приведены параметры, влияющие на архитектуру и алгоритмы работы системы синхронизации моделей. Предложен новый метод создания механизмов синхронизации, основанный на использовании процессных моделей, обоснована правомерность и удобство его использования для синтеза систем синхронизации, а также проверки их корректности. Поскольку устоявшейся методики создания систем синхронизации моделей в настоящий момент не выработано, проектирование подобных систем происходит с существенной завязкой на конкретную предметную область моделей методом проб и ошибок. Дальнейшая разработка предложенного подхода позволит создать мощный инструмент для определения структуры и верификации процесса синхронизации моделей, а также синтеза архитектуры систем синхронизации моделей

путем декларативного описания ее компонентов и анализа их взаимодействия друг с другом. Подобный инструмент в перспективе может стать основой для разработки систем синхронизации моделей из различных областей применения.

ЛИТЕРАТУРА

[1] Stevens P. Bidirectional Model Transformations in QVT: Semantic Issues and Open Questions. *Proc. of Int. Conf. on Model Driven Engineering Languages and Systems (MODELS'2007)*, vol. 4735 of LNCS, Springer, 2007, pp. 1–15.

[2] Stevens P. A Landscape of Bidirectional Model Transformations. *Int. Summer School, GTTSE 2007, Revised Papers*, vol. 5235 of LNCS, Springer, 2008, pp. 408–424.

[3] OMG. MOF2.0 query/view/transformation (QVT) adopted specification. OMG document ptc/05-11-01, 2005. URL: www.omg.org (дата обращения 10.04.2013).

[4] Foster J.N., Greenwald M.B., Moore J.T., Pierce B.C., Schmitt A. Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem. *ACM Transactions on Programming Languages and Systems* (TOPLAS), 2007, 29 (3).

[5] Mu S., Hu Z., Takeichi M. An algebraic approach to bidirectional updating. *APLAS*, 2004, vol. 3302 of LNCS, 2004.

[6] Xiong Y., Liu D., Hu Z., Zhao H., Takeichi M., Mei H. Towards automatic model synchronization from model transformations. *Proc. of the 22nd IEEE/ACM Int. Conf. on Automated software engineering, ACM*, 2007, pp. 164–173.

[7] Razavi A. *Incremental model synchronization*. Toronto, University of Waterloo, 2012.

[8] Razavi A., Kontogiannis K. *Incremental change management in models: challenges, alternatives and methodologies*. GRACE, 2008.

[9] Diskin Z. *Algebraic Models for Bidirectional Model Synchronization,* MODELS'2008, Toulouse, France, Springer, 2008.

[10] Ivkovic I., Kontogiannis K. Tracing evolution changes of software artifacts through model synchronization. *IEEE Int. Conf. on Software Maintenance (ICSM'04)* 1063-6773/04, 2004, pp. 252–261.

[11] Konigs A. Model transformation with triple graph grammars. *Proc. of the Workshop on Model Transformations in Practice*, MODELS'05, September 2005.

[12] Девятков В.В. Построение, оптимизация и модификация процессов. *Вестник МГТУ им. Н.Э. Баумана. Сер. Приборостроение,* 2012, № 2. с. 60–79.

[13] Миронов А.М. Теория процессов. URL: http://intsys.msu.ru/staff/mironov/processes.pdf (дата обращения 10.04.2013).

[14] Antkiewicz M., Czarnecki K. Design space of heterogeneous synchronization. *GTTSE,* Springer, 2007, vol. 5235 of LNCS, pp. 3–46.

[15] SWI PROLOG. URL: http://swi-prolog.org (дата обращения 25.05.2013).

Статья поступила в редакцию 24.06.2013

Ссылку на эту статью просим оформлять следующим образом:

Девятков В.В., Ошкало Д.В. Разработка процессов синхронизации моделей и принципов проверки их корректности. *Инженерный журнал: наука и инновации,* 2013, вып. 11. URL: http://engjournal.ru/catalog/it/hidden/1052.html

Девятков Владимир Валентинович родился в 1939 г., окончил Ленинградский институт точной механики и оптики в 1963 г. Д-р техн. наук, профессор, заведующий кафедрой «Информационные системы и телекоммуникации» МГТУ им. Н.Э. Баумана. Автор более 120 научных работ (из них 3 монографии). e-mail: deviatkov@ bmstu.ru

Ошкало Дмитрий Владиславович окончил МГТУ им. Н.Э. Баумана в 2012 г. Аспирант МГТУ им. Н.Э. Баумана по специальности «Информационные системы и телекоммуникации». Область научных исследований: моделеориентированная разработка программного обеспечения, мультиагентные системы. e-mail: dmitry.oshkalo@gmail.com