электронный журнал
МОЛОДЕЖНЫЙ НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК
Издатель ФГБОУ ВПО "МГТУ им. Н.Э. Баумана". Эл No. ФС77-51038.

УДК 004.413

Построение и анализ систем синхронизации моделей данных при помощи методов теории процессов

Ошкало Д.В., *аспирант*
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Информационные системы и телекоммуникации»
dmitry.oshkalo@gmail.com

Научный руководитель: Девятков В.В., д.т.н, профессор
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана
deviatkov@bmstu.ru

Введение

Моделе-ориентированный подход к разработке программного обеспечения основан на использовании формальных структурированных компонентов – моделей, участвующих в процессе построения различных артефактов – программного кода, схем баз данных, документации, конфигурационных настроек и т.д.

В то же время применение данного подхода требует организации процесса сопровождения моделей, направленного на внесение в них изменений и поддержание актуального состояния на всем протяжении работы над проектом. Зачастую между моделями существует семантическая взаимосвязь: они характеризуют с разных позиций один и тот же объект. В качестве примера можно привести различные виды моделей данных, с которыми работает создаваемая программа – реляционная модель, описывающая данные с точки зрения их хранения и объектно-ориентированная модель, представляющая данные с точки зрения их использования в коде программы. В общем случае возможна взаимосвязь между несколькими моделями.

Так как модели взаимосвязаны, при изменении одной из них, необходимо корректно отобразить изменения на другие модели. Данная задача носит название задачи синхронизации моделей, а реализующий ее процесс – синхронизацией моделей.

Задача синхронизации моделей при разработке программного обеспечения является одной из ключевых задач, стоящих перед разработчиками для обеспечения корректного и продуктивного процесса разработки. На практике выполнение данной активности занимает до 40% всего времени работы над проектом [1], и, таким образом, должно быть как можно более автоматизировано.

В настоящее время предлагается достаточно большое число методов решения задачи синхронизации моделей [2-5]. Однако, не рассматривается их применение в условиях многопользовательской среды разработки, т.е. данные подходы предлагают определенный механизм решения задачи синхронизации, но при этом игнорируется его функционирование в условиях параллельной работы над проектом нескольких разработчиков и проблемы, которые могут при этом возникнуть. В частности, не решаются такие вопросы как последовательность обеспечения доступа пользователей к моделям, корректность внесения изменений в модели, разрешение конфликтных ситуаций при обновлениях. Таким образом, проблема интеграции механизма синхронизации моделей в среду разработки, чтобы обеспечить эффективное его использование при работе над проектом, остается актуальной.

В качестве решения обозначенной проблемы в настоящей статье предлагается реализовать систему синхронизации моделей, имплементирующую некоторый механизм синхронизации, как отдельный модуль среды разработки. Для постановки задачи проектирования подобной системы необходимо рассмотреть и формально описать ее функциональные особенности и определить набор требований, которым она должна удовлетворять, а также принципы проверки системы на соответствие заявленным требованиям. Для решения этих задач предлагается использовать аппарат теории процессов.

Аппарат для описания системы

Аппарат теории процессов хорошо подходит для формального описания и анализа поведения систем, которые состоят из нескольких взаимодействующих компонентов, причем все эти компоненты работают параллельно, и взаимодействие компонентов происходит путём пересылки сигналов или сообщений от одних компонентов другим компонентам [6]. 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 именует 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 - для восприятий, !a – для внешних реакций, b – для внутренних реакций.

Нитью a^* будем называть кортеж (конечный или бесконечный) действий $a^*=a_0a_1a_2\ldots\ a_{m-2}a_{m-1}$. Выполнением нити процессом P называется последовательность выполнения ее действий в определенном порядке слева направо. 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.

Для описания систем взаимодействующих процессов введем следующие операции на процессах [7]:

- префиксное действие $a.P$;
- альтернативная композиция P_1+P_2;
- параллельная композиция $P_1|P_2$;

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

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 [7].

Формальное описание системы синхронизации и проверка свойств

В первом приближении система синхронизации моделей представляет собой систему параллельных процессов, работающих с несколькими взаимосвязанными некоторым образом моделями данных (рис. 1).

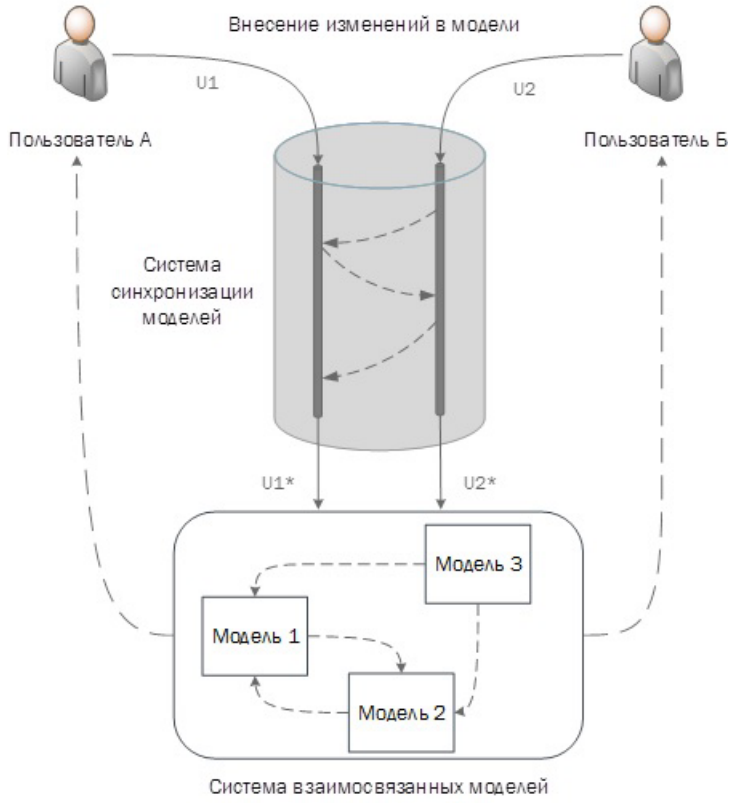

Рис. 1. Система синхронизации моделей

Система взаимосвязанных моделей используется совместно несколькими пользователями, которые вносят в нее изменения и получают информацию об изменениях, сделанных другими пользователями. Изменения, вносимые пользователями, при одновременном поступлении в систему анализируются для того, чтобы выявить, можно ли безопасно применить текущие наборы изменений или их следует модифицировать, после чего применить к моделям. При внесении изменений в одну из моделей, соответствующие изменения производятся в других связанных с ней моделях. После применения изменений и синхронизации пользователям становятся доступными обновленные модели.

Рассмотрим функциональную структуру системы синхронизации моделей, с которой одновременно работают *n* пользователей. Допустим, что работа пользователей с системой взаимосвязанных моделей состоит из двух операций: внесения изменений в одну или несколько моделей и обновление информации о состоянии системы моделей (рис. 2).

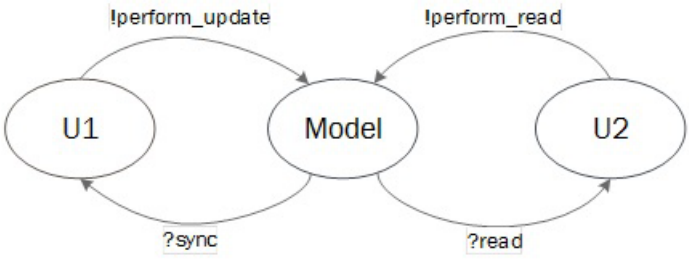

Рис. 2. Система синхронизации моделей как совокупность параллельных взаимодействующих процессов

При параллельной работе в подобной системе множества пользователей возможна следующая некорректная ситуация в поведении системы: конфликт одновременных обновлений и чтение ошибочного состояния моделей во время применения изменений к ним.

Допустим, чтобы избежать подобных ситуаций, используется механизм блокирования возможности обновления моделей до тех пор, пока не завершится текущая операция обновления или чтения. В этом случае, структура системы синхронизации моделей может быть формально представлена в виде параллельно работающих процессов *Model*, описывающего поведение системы моделей, и U_i, i=1..n, описывающих поведение пользователей:

$$Model = \sum_{i=1}^{n} \begin{pmatrix} ?upd\_req_i.!response_i.!sync_i.?release_i.Model + \\ +?read\_req_i.!response_i.!read_i.?release_i.Model \end{pmatrix}$$

$$U_i = !upd\_req_i.?response_i.!perform\_update_i.?sync_i.!confirm_i.!release_i.U_i + \\ +!read\_req_i.?response_i.!perform\_read.?read_i.!confirm_i.!release_i.U_i$$

$$System = Model \,|\, U_1 \,|\, .. \,|\, U_i \,|\, ... \,|\, U_n$$

Опишем действия, входящие в процессы.

Для системы моделей:

- ?*upd_req*$_i$ – получение системой моделей запроса на обновление от *i*-го пользователя;
- ?*read_req*$_i$ – получение системой моделей запроса на чтение данных от *i*-го пользователя;
- !*response*$_i$ – ответ *i*-му пользователю о возможности совершить запрошенное действие, установление блокировки;

- $!sync_i$ – синхронизация моделей и отправка результата *i*-му пользователю;
- $!read_i$ – отправка данных *i*-му пользователю;
- $?release_i$ – освобождение ресурсов *i*-м пользователем, снятие блокировки.

Для i-го пользователя:

- $!upd\_req_i$ – запрос на внесение изменений в модели;
- $!read\_req_i$ – запрос на чтение данных;
- $?response_i$ – получение ответа о возможности совершить запрошенное действие, установление блокировки;
- $!perform\_update_i$ – выполнение обновления;
- $!perform\_read_i$ – выполнение чтения данных;
- $!confirm$ – подтверждение окончания выполнения операции;
- $!release_i$ – освобождение ресурсов, снятие блокировки.

Продемонстрируем, что в построенной вышеописанным образом системе синхронизации действительно исключена возможность конфликта при одновременных обновлениях. В принятой системе обозначений данное свойство примет вид следующего процесса:

$$ConfUpdate = !\,perform.!\,confirm.ConfUpdate$$

Таким образом, если обновление моделей кем-либо из пользователей началось, никакие другие модификации не могут быть применены к моделям. Данное требование составляет спецификацию системы синхронизации моделей.

Построение доказательства утверждения о том, что процесс удовлетворяет своей спецификации основано на понятии эквивалентности процессов. Далее при доказательстве утверждений используются понятия сильной ($\sim$) и наблюдаемой ($\approx$) эквивалентности и наблюдаемой конгруэнции ($\approx^+$) [7].

Наличие данного свойства эквивалентно истинности отношения

$$System \approx ConfUpdate$$

Прежде чем приступать к доказательству соответствия системы своей спецификации, произведем некоторые преобразования процессов, описывающих систему. Так как механизм блокировки распространяется на обе операции обновления и чтения, данные операции представляют одно и то же поведение процесса пользователя, а также потому, что операции захвата и освобождения ресурсов одинаковы для всех

пользователей, к приведенным выше процессным выражениям можем применить следующую операцию переименования:

$$rename = \begin{pmatrix} confirm_i \to confirm \\ upd\_req_i \to req_i \\ read\_req_i \to req_i \\ perform\_update_i \to perform \\ perform\_read_i \to perform \\ read_i \to operate_i \\ sync_i \to operate_i \end{pmatrix}$$

Таким образом, процессные выражения для системы моделей и пользователей примут вид:

$$Model = \sum_{i=1}^{n} \begin{pmatrix} ?req_i.!response_i.!operate_i.?release_i.Model + \\ ?req_i.!response_i.!operate_i.?release_i.Model \end{pmatrix} \sim$$

$$\sim \sum_{i=1}^{n} ?req_i.!response_i.!operate_i.?release_i.Model$$

$$U_i = !req_i.?response_i.!perform.?operate_i.!confirm.!release_i.U_i +$$
$$+!req_i.?response_i.!perform.?operate_i.!confirm.!release_i.U_i \sim$$
$$\sim !req_i.?response_i.!perform.?operate_i.!confirm.!release_i.U_i$$

Далее преобразуем процесс *System*, используя теорему о разложении [7]:

$$System \sim \sum_{i=1}^{n} \tau.\begin{pmatrix} !response_i.!operate_i.?release_i.Model \,|\, U_1 \,|\, ... \,|\, ?response_i. \\ !perform.?operate_i.!confirm.!release_i.U_i \,|\, ... \,|\, U_n \end{pmatrix} \sim$$

$$\sim \sum_{i=1}^{n} \tau.\tau.\begin{pmatrix} !operate_i.?release_i.Model \,|\, U_1 \,|\, ... \\ ... \,|\, !perform.?operate_i.!confirm.!release_i.U_i \,|\, ... \,|\, U_n \end{pmatrix} \sim$$

$$\sim \sum_{i=1}^{n} \tau.\tau.!perform.\begin{pmatrix} !operate_i.?release_i.Model \,|\, U_1 \,|\, ... \\ ... \,|\, ?operate_i.!confirm.!release_i.U_i \,|\, ... \,|\, U_n \end{pmatrix} \sim$$

$$\sim \sum_{i=1}^{n} \tau.\tau.!perform.\tau.\begin{pmatrix} ?release_i.Model \,|\, U_1 \,|\, ... \\ ... \,|\, !confirm.!release_i.U_i \,|\, ... \,|\, U_n \end{pmatrix} \sim$$

$$\sim \sum_{i=1}^{n} \tau.\tau.!perform.\tau.!confirm.\begin{pmatrix} ?release_i.Model \,|\, U_1 \,|\, ... \\ ... \,|\, !release_i.U_i \,|\, ... \,|\, U_n \end{pmatrix} \sim$$

$$\sim \sum_{i=1}^{n} \tau.\tau.!perform.\tau.!confirm.\tau.\begin{pmatrix} Model \,|\, U_1 \,|\, ... \\ ... \,|\, U_i \,|\, ... \,|\, U_n \end{pmatrix} =$$

$$= \sum_{i=1}^{n} \tau.\tau.!perform.\tau.!confirm.\tau.System$$

Применив свойства процессов [7]:

$$PP + P \sim P$$

$$\alpha.\tau.P \approx^{+} \alpha.P\text{'}$$

получим:

$$System \approx^{+} \tau.!\, perform.!\, confirm.System$$

Анализируя проверяемое свойство, заметим, что

$$\tau.ConfUpdate \approx^{+} \tau.!\, perform.!\, confirm.(\tau.ConfUpdate)$$

Таким образом, получаем

$$System \approx^{+} \tau.ConfUpdate\,,$$

откуда следует, что [7]

$$System \approx ConfUpdate$$

Таким образом, формально доказано, что примененный механизм блокировки действий пользователей гарантирует отсутствие перечисленных выше конфликтных ситуаций при работе системы синхронизации моделей.

Анализ свойств системы

Заметим, что при рассмотрении особенностей поведения системы не принимались в расчет данные, с которыми она работает. Однако, поведенческие свойства системы синхронизации могут зависеть от типов данных.

Допустим, перед внесением изменений необходимо подтвердить их корректность.

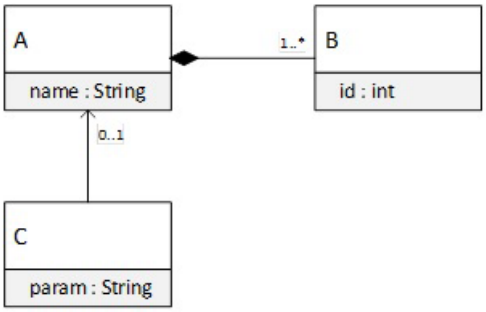

Рис. 3. UML-диаграмма модели данных

Например, для модели данных, описываемой UML-диаграммой, изображенной на рис. 3, корректные изменения будут включать:

- создание экземпляра класса A, связанного с одним создаваемым экземпляром класса B;
- добавление нового экземпляра класса B к существующему экземпляру класса A;
- создание/изменение/удаление экземпляра класса C;
- добавление к существующему экземпляру класса C ссылки на существующий экземпляр класса A;
- удаление экземпляра класса A вместе со всеми принадлежащими ему экземплярами класса B;
- изменение параметров экземпляра класса A;
- изменение параметров экземпляра класса B;
- удаление экземпляра класса B, если он не последний среди принадлежащих экземпляру класса A.

Любые другие изменения, например, создание экземпляра класса A без, по крайней мере, одного экземпляра класса B являются некорректными.

Для описания свойств системы, содержащих информацию о данных, выразительных средств использованной в данной статье теории процессов недостаточно. На практике при возникновении подобных ситуаций применяют расширение некой базисной алгебры процессов с тем, чтобы в новой алгебраической системе выразить понятия, на основании которых можно было бы выражать интересующие аспекты поведения системы (например, временные параметры или работу с определенными типами данных) [8-10].

Таким образом, в описанном выше случае необходимо расширить аппарат используемой теории процессов возможностью выражать свойства моделей данных, с которыми работают процессы, и отношения между ними.

Заключение

В статье рассмотрена задача проектирования системы синхронизации моделей как отдельного модуля среды разработки. Приведен пример формального описания системы, отражающее ее поведенческие особенности и не зависящее от используемого механизма синхронизации моделей. Составлена спецификация системы и рассмотрен пример

доказательства соответствия поведения системы заявленным требованиям. Отмечена необходимость дополнения примененного для верификации аппарата возможностью использования данных, с которыми работает система. Дальнейшая разработка позволит создавать более точные спецификации систем синхронизации данных.

Список литературы

1. Haan J.D. 15 reasons why you should start using Model Driven Development. Available at: http://www.theenterprisearchitect.eu/blog/2009/11/25/15-reasons-why-you-should-start-using-model-driven-development , acessed: 20.11.2014.
2. Leblebici E., Anjorin A., Schurr A., Hildebrandt S., Rieke J., Greenyer J. A Comparison of Incremental Triple Graph Grammar Tools. 13th International Workshop on Graph Transformation and Visual Modeling Techniques: proceedings (Grenoble, April, 05-06 2014). Grenoble, 2014. Vol. 67. P.1-14.
3. Diskin, Z., Algebraic Models for Bidirectional Model Synchronization // 11th International Conference (Toulouse, June, 3-6 2008): proceedings. Toulouse, 2008. P.21-36.
4. Antkiewicz M. and Czarnecki K. Design space of heterogeneous synchronization. In Generative and Transformational Techniques in Software Engineering (Berlin, July, 07-11 2008): proceedings. Berlin, 2008. P.3-46.
5. OMG. MOF2.0 query/view/transformation (QVT) adopted specification. OMG document ptc/05-11-01, 2005. Available at: http://www.omg.org/, accessed: 20.11.2014.
6. Девятков В.В. Построение, оптимизация и модификация процессов // Вестник МГТУ им. Н.Э.Баумана. Сер. Приборостроение. 2012. № 2. С. 60-79.
7. Миронов, А.М. Теория процессов. Режим доступа: http://intsys.msu.ru/staff/mironov/processes.pdf (дата обращения: 20.11.2014).
8. Baeten J.C.M. A brief history of process algebra // Theoretical Computer Science. 2005. Vol. 335. Issue 2-3. P. 131-146.
9. Glabbeek, R. J. van. Notes on the methodology of CCS and CSP // Theoretical Computer Science archive. 1997. Vol. 177. P. 329 – 349.
10. mCRL. A language and tool set to study communicating processes with data. Available at: http://homepages.cwi.nl/~mcrl/, accessed: 20.11.2014.