УДК 004.415

П.Д. Дробинцев, И.В. Никифоров, Н.В. Воинов, В.П. Котляров

ПОДХОД К ТЕСТИРОВАНИЮ ПАРАЛЛЕЛЬНЫХ СИСТЕМ НА ОСНОВЕ UCM-СПЕЦИФИКАЦИЙ

P.D. Drobintsev, I.V. Nikiforov, N.V. Voinov, V.P. Kotlyarov

APPROACH TO CONCURRENT SYSTEMS TESTING BASED ON UCM SPECIFICATION

Изучен подход к тестированию параллельных программных систем, основанный на использовании спецификаций на языке UCM. Рассмотрены UCM-конструкции, позволяющие задавать многопоточное поведение в системе. Описаны эквивалентные им конструкции на языке базовых протоколов, с которым работает средство генерации тестовых сценариев. Представлен подход к автоматическому созданию тестов для многопоточных систем в виде MSC-диаграмм. Представленный подход позволяет на основе анализа независимости параллельных потоков автоматически получать корректные наборы тестов для тестирования параллельных распределенных систем.

ФОРМАЛЬНАЯ МОДЕЛЬ; СИНХРОНИЗАЦИЯ ПРОЦЕССОВ; ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ.

The current paper presents an approach to testing of concurrent program systems based on UCM specifications usage. We have studied UCM constructions specifying multithreading system behavior. We have also described equivalent constructions using the language of basic protocols that is operated by test scenario generator. The approach to automated creation of test scenarios in MSC notation for multithreading systems has been shown. The described approach based on the analysis of concurrent threads allows us to obtain automatically verified test suites for testing concurrent distributed systems.

FORMAL MODEL; THREAD SYNCHRONIZATION; PARALLEL COMPUTING.

В настоящее время в силу активного развития компьютерных технологий и повсеместной информатизации общества огромное количество задач решается с помощью программно-аппаратных комплексов, производящих распределенные вычисления. Распределенные вычисления – это способ решения трудоемких вычислительных задач с использованием нескольких компьютеров, чаще всего объединенных в параллельную вычислительную систему [1]. При этом параллельная вычислительная система представляет собой физическую компьютерную или программную систему, реализующую тем или иным способом параллельную обработку данных на многих вычислительных узлах [2]. Сложность подобных программных систем и высокие требования к их качеству делают все более актуальной задачу их тестирования.

Для многих приложений использование параллельных процессов решает задачу эффективности, т. е. получения результатов вычислений в рамках заданных временных ограничений. Однако тестирование программной системы на основе использования параллельных процессов сопряжено с проблемой взрыва состояний, порождаемого необходимостью проверки состояний всех процессов системы во время прогона теста [8].

В настоящей статье описывается подход, позволяющий решить данную проблему для определенного класса систем, требования на которые представлены в виде формальной модели на языке спецификаций высокого уровня UCM (Use Case Maps) [3].

Моделирование параллельного исполнения в UCM

Язык UCM позволяет создавать модели

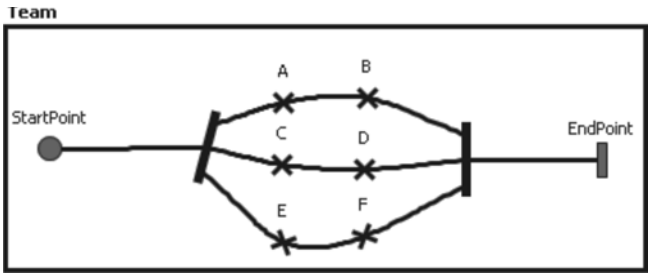

Рис. 1. Порождение и синхронизация процессов

систем, использующих параллельные потоки. Для порождения и последующей синхронизации потоков используются специальные элементы: AndFork – для порождения потоков и AndJoin – для их синхронизации. Следует отметить, что при создании модели с параллельным поведением необходимо соблюдать ограничения на создание подобных конструкций, которые позволят получить синтаксически корректную модель [4]. На рис. 1 приведена диаграмма, демонстрирующая применение элементов AndFork и AndJoin для создания и синхронизации трех параллельных потоков.

Особенностью разработки многопоточных UCM-моделей является интерпретация компонентов, являющихся элементами языка UCM, процессами, а параллельных ветвей, заданных элементами AndFork и AndJoin, – потоками внутри процесса. Все процессы в системе параллельны друг другу и существуют с начала исполнения сценариев, в то время как потоки внутри компонентов ограничены по времени жизни и не могут превышать время жизни процесса-родителя.

Создавать параллельные потоки на пути позволяет элемент AndFork. Каждая исходящая ветка соответствует параллельному потоку. В соответствии с семантикой UCM в отличие от элемента OrFork, позволяющего определять альтернативное поведение системы, на исходящих ветвях AndFork нельзя задавать условия ветвления, т. е. при достижении элемента AndFork все исходящие ветки могут начать свое параллельное исполнение.

На основе формального представления UCM-модели может быть проведена автоматическая генерация модели в нотации базовых протоколов [5]. Подобный транслятор реализован и описан в рамках работ над созданием интегрированной технологии для анализа и верификации спецификаций [6].

Рассмотрим некоторые детали работы транслятора. При работе с параллельным поведением каждая исходящая ветка формализуется базовым протоколом, представляющим собой тройку Хоара, вида $\forall x(\alpha \rightarrow < u > \beta)$, где x – список (типизированных) параметров, α и β – формулы базового языка, u – процесс протокола (вполне определенное поведение) [5]. При этом в каждом из базовых протоколов генерируется уникальный для ветки поток управления (рис. 2).

На диаграмме рис. 1 происходит гене-

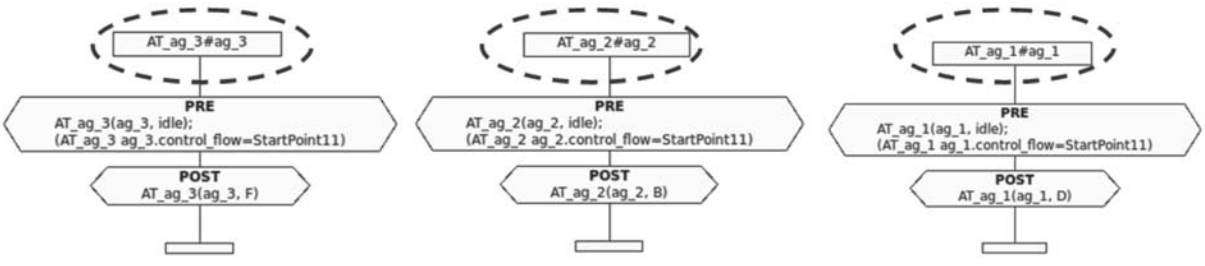

Рис. 2. Формализация параллельного поведения в виде базовых протоколов

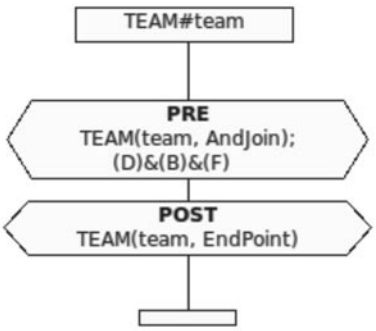

Рис. 3. Протокол синхронизации

рация трех потоков с тремя уникальными агентами. На рис. 2 уникальные агенты для потоков отмечены пунктирными овалами.

Конкретный поток управления используется во всех протоколах соответствующей ветви параллельного исполнения, что позволяет разделить множество протоколов по различным ветвям описания параллельного поведения.

Стоит отметить, что агенты, отвечающие за исполнение потока управления, содержатся в родительском потоке. Эта информация отображается в файле описания окружения формальной модели.

Элемент AndJoin в свою очередь позволяет синхронизовать входящие в него параллельные ветви. Для синхронизации необходимо, чтобы все входящие потоки завершили свое исполнение, после чего общий поток, определяемый как поток родитель для синхронизуемых потоков, продолжает свое исполнение. Элемент формализуется одним базовым протоколом (рис. 3).

На рис. 3 видно, что в базовом протоколе присутствует условие синхронизации – B&D&E, которое описывает необходимость завершения всех потоков до точки синхронизации и одновременно является условием применимости данного протокола.

Использование описанных правил преобразования элементов, задающих параллельное поведение на UCM-диаграмме, позволяет реализовать переход от UCM-диаграмм, описывающих многопоточное поведение, к формальной модели на языке базовых протоколов, сохраняя при этом семантику исходной диаграммы, что в свою очередь предоставляет возможность проведения ее формальной верификации. Поведенческие сценарии или трассы, полученные на инструментальной системе VRS [5] в процессе символьной верификации поведенческой модели приложения, могут использоваться для его тестирования.

Подход к генерации тестовых сценариев для параллельных процессов

Одной из основных проблем при тестировании параллельных систем является проблема взрыва состояний, вызванная интерливингом между взаимодействующими параллельными потоками. Количество возможных состояний параллельных потоков, подлежащих проверке при тестировании, растет экспоненциально в зависимости от количества потоков и состояний каждого потока. Тестирование подобных систем чрезвычайно затруднительно в силу некоторых обстоятельств.

Во-первых, при тестировании возникает необходимость полного перебора всех возможных комбинаций состояний, в которых может оказаться тестируемая система, и соответственно путей, приводящих ее в это состояние.

Во-вторых, сам по себе перевод системы в необходимое состояние при наличии нескольких параллельных потоков является нетривиальной задачей, т. к. исполнение одного потока может оказывать влияние на исполнение другого, что приводит к необходимости управления каждом потоком в отдельности и анализа их взаимного влияния.

В случае если известно, что параллельные потоки независимы, то задача генерации тестовой трассы решается достаточно просто. Необходимо запомнить состояние модели перед элементом AndFork, выделить в тестируемой системе параллельные потоки (от AndFork до AndJoin) и вставить их в тестовый сценарий последовательно: один за другим, восстанавливая запомненное состояние перед началом каждой нити. В итоге после верификации получим корректный тестовый сценарий без применения интерливинга. Если параллельные потоки зависимы, то предварительно в ре-

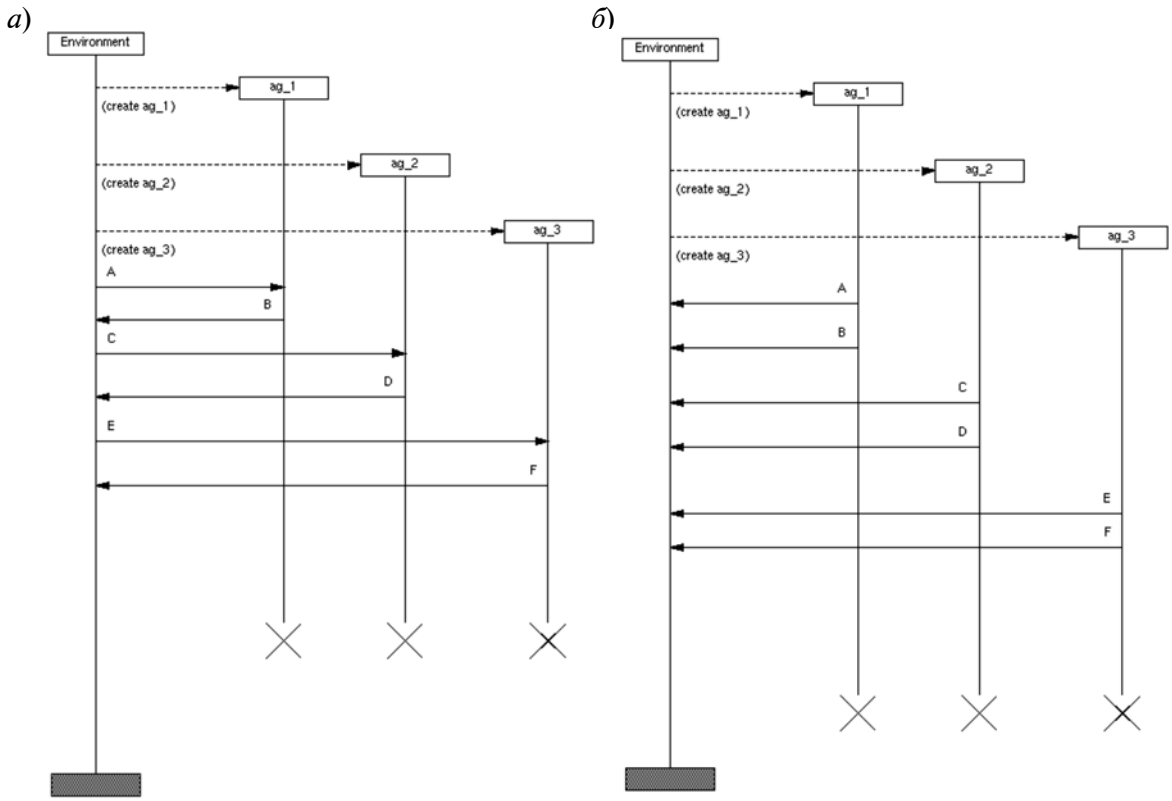

Рис. 4. Тесты для параллельных потоков

зультате статического анализа необходимо найти точки взаимозависимости и обеспечить синхронизацию зависимых взаимодействий, за счет чего скорректированные параллельные потоки станут независимыми от изменения последовательности воздействий на систему, что позволит свести задачу генерации тестовых трасс параллельных потоков к предыдущей.

Следует отметить, что даже при условии независимости параллельных потоков для проведения полноценного тестирования недостаточно использования одного теста. Это связано с тем, что нередко в системах с параллельными потоками управление каждым потоком с точным предсказанием времени его работы невозможно, и таким образом возникает необходимость применения нелинейных тестов. Проиллюстрируем данную проблему на примере.

Предположим, что потоки системы, изображенной на рис. 1, независимы, а элементы responsibility описывают прием и посылку пар сигналов (A; B), (C; D) и (D; E) соответственно. Тогда тест, основанный на предположении о независимости потоков, может выглядеть следующим образом (рис. 4 *а*).

Для описания теста использован язык MSC (Message Sequence Charts) [6]. Из рис. 4 *а* видно, что единственная правильная последовательность поведения системы – это последовательность посылок и приемов сигналов A, B, C, D, E, F, и в данном случае она будет корректна, т. к. тест выступает инициатором обмена сообщениями с системой. Однако если предположить, что responsibility описывают прием пар сигналов (A; B), (C; D) и (E; F) от тестируемой системы, то полученный тест на рис. 4 *б* будет некорректным, т. е. будет завершаться с ошибкой даже в случае корректного поведения системы. Проблема в данном случае связана с тем, что при тестировании реального программного обеспечения невозможно без специального управления со стороны теста гарантировать, что пара сигналов (A; B) придет первой (то же утверждение

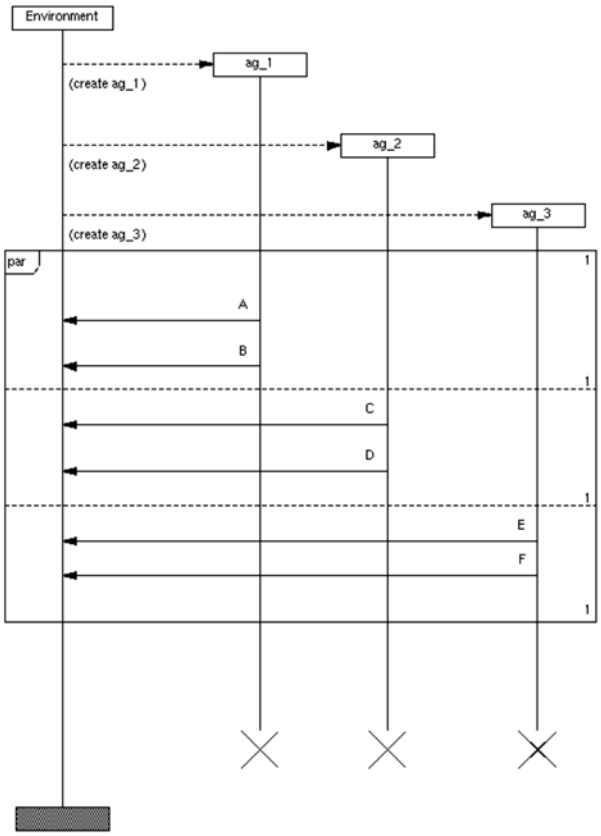

Рис. 5. Использование нелинейных конструкций языка MSC

касается остальных пар сигналов). Данная проблема носит название *гонки сигналов*.

Для решения данной проблемы может использоваться оператор par, являющийся синтаксической конструкцией языка MSC и позволяющий описывать взаимодействие параллельных процессов. На рис. 5 приведен корректный тест, полученный добавлением оператора par.

Из рисунка видно, что оператор состоит из трех блоков, каждый из которых описывает взаимодействие окружения с одним из параллельных потоков. Подобная запись определяет последовательность прихода сигналов в рамках одного конкретного потока и дает возможность полного перебора (без явного указания всех возможных вариантов, что существенно сокращает запись теста) между параллельными процессами. При этом не происходит генерации тестов для описания каждого конкретного перебора и контроль правильности работы системы осуществляется в рамках одного теста с указанным оператором. Следует отметить, что добавление оператора par является простой процедурой и может быть полностью автоматизировано на основе информации о порождении и синхронизации потоков.

Таким образом, представленный подход позволяет на основе анализа независимости параллельных потоков автоматически получать корректные наборы тестов для тестирования параллельных распределенных систем.

СПИСОК ЛИТЕРАТУРЫ

1. **Таненбаум Э.С., ван Стеен М.** Распределенные системы. Принципы и парадигмы. СПб.: Питер, 2003. 877 с.

2. **Almasi G.S., Gottlieb A.** Highly Parallel Computing. Benjamin-Cummings publishers, Redwood City, CA, 1989.

3. ITU-T Recommendation Z.151. User requirements notation (URN), 10.2012 [электронный ресурс].

4. **Баранов С.Н., Дробинцев П.Д., Летичевский А.А., Котляров В.П.** Верификационная технология базовых протоколов для разработки и проектирования программного обеспечения // Программные продукты и системы. 2005. № 1(69). С. 25–28.

5. **Ануреев И.С., Баранов С.Н., Белоглазов Д.М., Бодин Е.В., Дробинцев П.Д., Колчин А.В., Котляров В.П., Летичевский А.А., Летичевский А.А. мл., Непомнящий В.А., Никифоров И.В., Потиенко С.В., Прийма Л.В., Тютин Б.В.** Средства поддержки интегрированной технологии для анализа и верификации спецификаций телекоммуникационных приложений // Труды СПИИ РАН. 2013. № 3(26). С. 349–384.

6. ITU Recommendation Z.120. Message Sequence Charts (MSC), 11.1999 [электронный ресурс].

7. **Никифоров И.В., Котляров В.П., Дробинцев П.Д.** Ограничения на многопоточные конструкции и временные задержки языка UCM // Научно-технические ведомости СПбГПУ. Информатика. Телекоммуникации. Управление. СПб.: Изд-во СПбГПУ, 2013. № 3(174). С. 148–153.

8. **Бурдонов И.Б., Грошев С.Н., Демаков А.В., Камкин А.С., Косачев А.С., Сортов А.А.** Параллельное тестирование больших автоматных моделей // Информационные технологии. Вестник Нижегородского университета им. Н.И. Лобачевского. 2011. № 3(2). С. 187–193.

REFERENCES

1. **Tanenbaum A.S., van Steen M.** *Raspredelennyye sistemy. Printsipy i paradigmy* [*Destributed systems. Principles and paradigms*]. St. Petersburg: Piter Publ., 2003, 877 p. (rus)

2. **Almasi G.S., Gottlieb A.** *Highly Parallel Computing.* Benjamin-Cummings publishers, Redwood City, CA, 1989.

3. *ITU-T Recommendation Z.151. User requirements notation (URN),* 10.2012.

4. **Baranov S.N., Drobintsev P.D., Letichevskiy A.A., Kotlyarov V.P.** Verifikatsionnaya tekhnologiya bazovykh protokolov dlya razrabotki i proyektirovaniya programmnogo obespecheniya, *Programmnyye produkty i sistemy*, 2005, No. 1(69), Pp. 25–28. (rus)

5. **Anureyev I.S., Baranov S.N., Beloglazov D.M., Bodin Ye.V., Drobintsev P.D., Kolchin A.V., Kotlyarov V.P., Letichevskiy A.A., Letichevskiy A.A. ml., Nepomnyashchiy V.A., Nikiforov I.V., Potiyenko S.V., Priyma L.V., Tyutin B.V.** Sredstva podderzhki integrirovannoy tekhnologii dlya analiza i verifikatsii spetsifikatsiy telekommunikatsionnykh prilozheniy, *Trudy SPII RAN*, 2013, No. 3(26), Pp. 349–384. (rus)

6. *ITU Recommendation Z.120. Message Sequence Charts (MSC).* 11.1999.

7. **Nikiforov I.V., Kotlyarov V.P., Drobintsev P.D.** Ogranicheniya na mnogopotochnyye konstruktsii i vremennyye zaderzhki yazyka UCM, *Nauchno-tekhnicheskiye vedomosti SPbGPU. Informatika. Telecommunikatcii. Upravlenie.* St. Petersburg: SPbGPU Publ., 2013, No. 3(174), Pp. 148–153. (rus)

ДРОБИНЦЕВ Павел Дмитриевич – *доцент кафедры информационных и управляющих систем Санкт-Петербургского государственного политехнического университета, кандидат технических наук.*

195251, Россия, Санкт-Петербург, ул. Политехническая, д. 29.
E-mail: drob@ics2.ecd.spbstu.ru

DROBINTSEV, Pavel D. *St. Petersburg State Polytechnical University.*
195251, Politekhnicheskaya Str. 29, St. Petersburg, Russia.
E-mail: drob@ics2.ecd.spbstu.ru

НИКИФОРОВ Игорь Валерьевич – *ассистент кафедры информационных и управляющих систем Санкт-Петербургского государственного политехнического университета, кандидат технических наук.*

195251, Россия, Санкт-Петербург, ул. Политехническая, д. 29.
E-mail: igor.nikiforovv@gmail.com

NIKIFOROV, Igor V. *St. Petersburg State Polytechnical University.*
195251, Politekhnicheskaya Str. 29, St. Petersburg, Russia.
E-mail: igor.nikiforovv@gmail.com

ВОИНОВ Никита Владимирович – *доцент кафедры информационных и управляющих систем Санкт-Петербургского государственного политехнического университета, кандидат технических наук.*
195251, Россия, Санкт-Петербург, ул. Политехническая, д. 29.
E-mail: voinov@ics2.ecd.spbstu.ru

VOINOV, Nikita V. *St. Petersburg State Polytechnical University.*
195251, Politekhnicheskaya Str. 29, St. Petersburg, Russia.
E-mail: voinov@ics2.ecd.spbstu.ru

КОТЛЯРОВ Всеволод Павлович – *профессор кафедры информационных и управляющих систем Санкт-Петербургского государственного политехнического университета, кандидат технических наук.*
195251, Россия, Санкт-Петербург, ул. Политехническая, д. 29.
E-mail: vpk@spbstu.ru

KOTLYAROV, Vsevolod P. *St. Petersburg State Polytechnical University.*
195251, Politekhnicheskaya Str. 29, St. Petersburg, Russia.
E-mail: vpk@spbstu.ru