Исполнение моделей при помощи виртуальной машины

К.В. Буздин

1. Для чего нужны модели

Прошло уже почти 20 лет со времени появления в 1986 году вызвавшей множество споров статьи Фредерика Брукса «Серебряной пули нет». В данной работе автор предрекал, что в течение ближайшего десятилетия с момента её выхода не возникнет методов, использование которых позволит на порядок величин повысить производительность разработки программного обеспечения. Эта статья вызвала множество споров и работ с опровержениями, и в 1995 году Ф. Брукс опубликовал ответы на некоторые из критических замечаний [1].

По признанию самого Ф. Брукса, наиболее тщательный анализ его статьи был предпринят Дэвидом Харелом в работе «Кусая серебряную пулю» [2]. Д. Харел рассматривал «Серебряной пули нет» как чрезмерно пессимистическую и намеревался высветить в своей работе более яркую сторону проблемы, предпосылая статье подзаголовок «К светлому будущему программных разработок». Им была предложена унифицированная система разработки программного обеспечения, главной целью которой было освободить программиста от необходимости рассуждать на чрезмерно глубоком уровне детализации, предоставить возможность размышлять над решением проблемы и представлять свои идеи при помощи соответствующей высокоуровневой нотации, а именно моделей. «Использование подходящего графического формализма может оказать впечатляющий эффект на инженеров и программистов» [2]. Кроме самих моделей для эффективной работы необходимы средства для их изучения и анализа, среди которых одной из наиболее полезных, по мнению Д. Харела, является возможность исполнения модели.

В настоящее время всё чаще упоминается метод разработки на основе моделей, или Model Driven Development (MDD) [5], обещающий стать первым технологическим скачком со времён появления компиляторов, который сможет значительно ускорить процесс создания программного обеспечения и улучшить его качество. Лидирующие позиции в данной области призван занять стандарт Model Driven Architecture (MDA), разработкой которого занимается консорциум OMG. Разработка в рамках MDA предполагает, что сначала создаётся одна или несколько абстрактных моделей системы, которые далее проходят несколько стадий автоматизированной трансформации в более детальные модели и, в конечном итоге, преобразуются в код на языке программирования. Как проще написать одну строку на Java, чем 10 строк на языке ассемблера, так же проще построить графическую модель на языке UML, чем писать на Java, считают сторонники подхода MDA. Но для успешного применения моделей необходима не только удобная нотация для их записи, которую предоставляет язык UML, но и соответствующий инструментарий для их изучения.

Автоматизация разработки является тем технологическим средством, которое должно привести к ускорению процесса создания программного обеспечения и повышению его качества. Тем не менее, ранние попытки применения автоматизации в сфере моделирования ограничивались лишь поддержкой в создании диаграмм и генерацией скелетного кода, что было недостаточным для того, чтобы существенно увеличить производительность разработки. Наибольшей же отдачи от MDD можно добиться лишь после полной реализации его потенциальных возможностей для автоматизации, которые включают:

- автоматическую генерацию полного программного кода по моделям (а не скелетного кода или отдельных фрагментов),
- автоматическую верификацию моделей (например, при помощи их исполнения).

Данная статья посвящена теме исполнения моделей. Автоматическая верификация подразумевает возможность при помощи компьютера определить, удовлетворяет ли модель требованиям, предъявляемым к системе. Такая проверка может принимать различные формы, включая формальный математический вывод, но наиболее часто под этим подразумевается тестирование и отладка моделей путём их исполнения. В любом случае важно иметь такую возможность для моделей, находящихся на высоком уровне абстракции и даже неполных, которые появляются на ранних стадиях процесса разработки, поскольку именно тогда и принимается большинство фундаментальных решений.

2. Почему важно, чтобы модели были исполняемыми

Одним из наиболее важных способов нашего познания является обучение посредством эксперимента. В контексте данной статьи это означает исполнение моделей. Дэвид Харел сравнивает модели, которые не могут быть исполнены, с машинами, у которых нет двигателя. Важным преимуществом исполняемых моделей является то, что они на ранних стадиях обеспечивают возможность получения опыта работы с создаваемой системой. В данном случае уместна аналогия с языками программирования. Когда мы изучаем новый язык, мы всегда бываем очень воодушевлены успешным выполнением первой тривиальной программы “Hello, world!”. Этот простой опыт укрепляет нашу уверенность в правильности действий и служит отправной точкой в дальнейших исследованиях. Информация, полученная в результате экспериментов, помогает перейти от формального знания к пониманию. Путём исполнения модели, отражающей наши ожидания относительно поведения системы,

задолго до собственно реализации самой системы мы можем убедиться, что она действительно будет работать правильно. В случае, если при этом будут обнаружены отклонения от ожидаемого поведения, мы можем вернуться к модели, изменить её и запустить тот же сценарий, на котором были выявлены недостатки. Таким образом, исполнение модели позволяет найти ошибки, которые иначе остались бы незамеченными до тех пор, пока большая часть системы уже не была бы реализована, и было бы слишком поздно. И это начинает проявляться сразу же после того, как модель впервые будет запущена на исполнение.

Кроме того, на ранних стадиях создания системы к работе могут привлекаться представители заказчика. В этом случае исполняемая модель будет играть роль прототипа при уточнении требований.

3. Результат исполнения модели

Результаты, получаемые при исполнении модели, могут быть подразделены на два вида:

- получаемые непосредственно в процессе исполнения;
- получаемые по завершении исполнения.

В первом случае исполнение модели предоставляет возможность получить опыт от общения с реальной системой. При этом допустимо пошаговое исполнение с целью более детальной отладки. При интерактивном исполнении пользователь играет роль окружения системы, он может порождать события, наблюдать и изменять значения переменных, выбирать путь дальнейшего продолжения исполнения, если имеется несколько возможностей. В ответ инструмент поддержки исполнения должен перевести систему в новое состояние. Кроме того, поскольку большинство моделей имеют графическое представление, то изменение состояния должно отражаться на самой модели, например, путём изменения цвета некоторых её элементов. Для получения эффекта непрерывного исполнения интересующую последовательность событий из окружения системы можно подготовить заранее. В таком случае, при наличии соответствующей графической поддержки, можно наблюдать анимацию модели. Аналогичного эффекта можно добиться, если инструмент сам будет генерировать события случайным образом. Также полезна возможность использования точек останова для изучения конкретных состояний моделируемой системы. Особый интерес представляет полное исполнение модели, когда инструмент генерирует все возможные сочетания событий из окружения с целью проверки всех возможных вариантов поведения моделируемой системы. При этом необходима возможность формального описания интересующих разработчика ситуаций и автоматического их распознавания, поскольку результат полного исполнения для реальных систем будет слишком объёмен для исследования его человеком.

В процессе исполнения модели инструмент может собирать различные данные, например, подсчитывать некоторые метрики или составлять трассы, которые представляют собой последовательность событий, возникающих в системе.

Подобные данные могут оказаться полезными в дальнейшей работе над системой, например для создания тестов.

4. Способы исполнения моделей

Рассмотрим следующие способы исполнения моделей:

- непосредственная интерпретация,
- генерация автоматной модели и её исполнение,
- использование виртуальной машины.

4.1. Интерпретация

В данном случае для исполнения модели создаётся специальная программа, называемая интерпретатором. На вход интерпретатору подаётся сама модель в том виде, в каком она была создана пользователем, без каких либо промежуточных преобразований. Основным недостатком интерпретации является её низкая скорость.

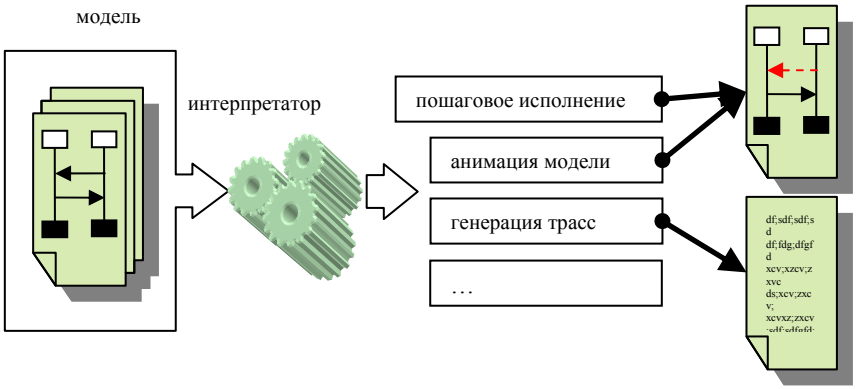

Рис. 1. Интерпретация

На рисунке 1 представлена схема исполнения моделей с использованием непосредственной интерпретации.

4.2. Генерация автоматной модели

Многие современные реализации исполняемых моделей базируются на синтезе автоматов. Далее конечные автоматы можно интерпретировать непосредственно или использовать для последующей генерации кода. Применение автоматов в данном случае обусловлено следующими их преимуществами: они представляются в хорошо формализованном виде, имеется развитый математический аппарат для работы с конечными автоматами, который позволяет проводить над ними формальные исследования и преобразования, и, наконец, автоматные модели довольно хорошо отображаются в код на целевых языках.

Но важно подчеркнуть, что хотя автоматные модели являются хорошо формализованным и мощным средством спецификации, они могут оказаться сложнее в работе и менее наглядными для неподготовленных пользователей, чем другие модели. Кроме того, при рассмотрении программной системы возможен разрыв, связанный с различиями в степени абстракции, используемой в автоматных и других спецификациях.

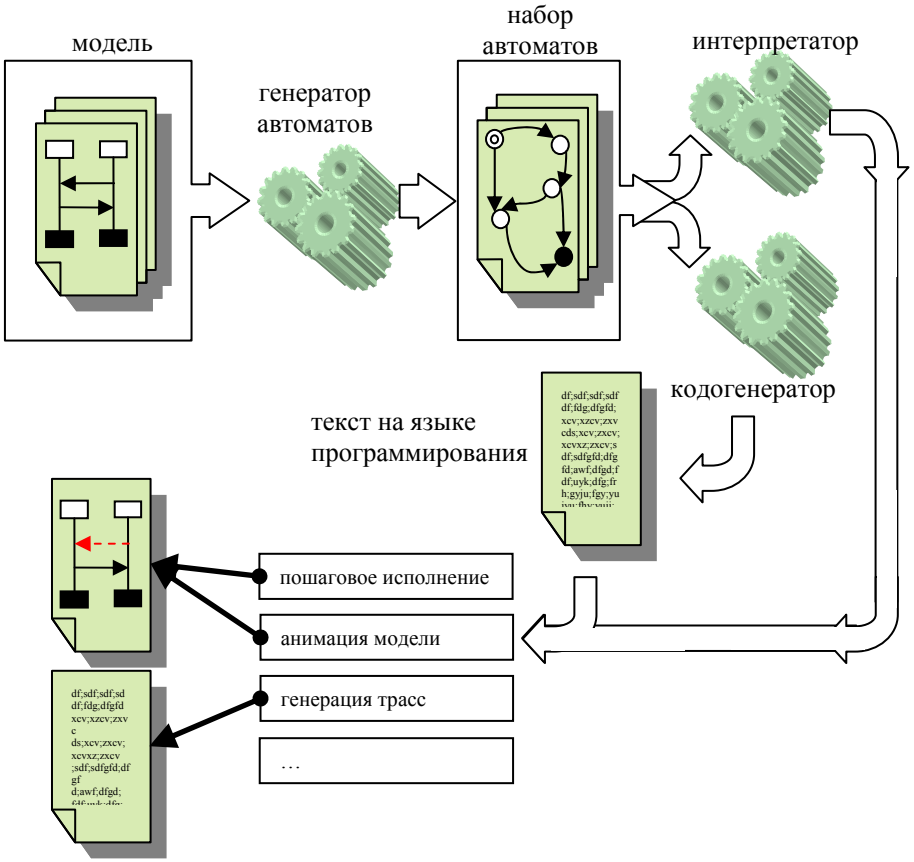

Рис. 2. Использование автоматов

Следует отметить, что при исполнении моделей при помощи автоматов появляются дополнительные трудности при реализации визуализации исполнения. Дело в том, что исполнению подвергается уже не сама модель, а её представление в виде автоматов или даже код на языке программирования, а отображать изменения необходимо в терминах самой модели. Таким образом, по сравнению с непосредственной интерпретацией моделей, в данном случае необходимо прилагать дополнительные усилия для определения соответствия конструкций, употребляемых в моделях, и их образов в автоматах и коде.

4.3. Виртуальная машина

В качестве промежуточного представления для исполняемых моделей могут использоваться не только конечные автоматы. Дело в том, что они представляют собой слишком общий и низкоуровневый инструмент. Конечные автоматы подходят для решения очень широкого круга задач, поэтому в них может не оказаться инструментов, специфичных для выбранных для исполнения моделей. Кроме того, сложность конечных автоматов может резко возрастать при увеличении сложности самой описываемой ими модели [4].

Альтернативной основой для реализации исполняемых моделей может послужить виртуальная машина. Виртуальная машина — это абстрактная машина, для которой существует интерпретатор. В свою очередь, абстрактная машина — это модель процессора, которая не предназначена для реализации «в железе». Для неё определены набор команд и модель памяти. В памяти хранятся данные, с которыми может работать машина. В сравнении с автоматами при создании виртуальной машины имеется большая свобода выбора модели данных.

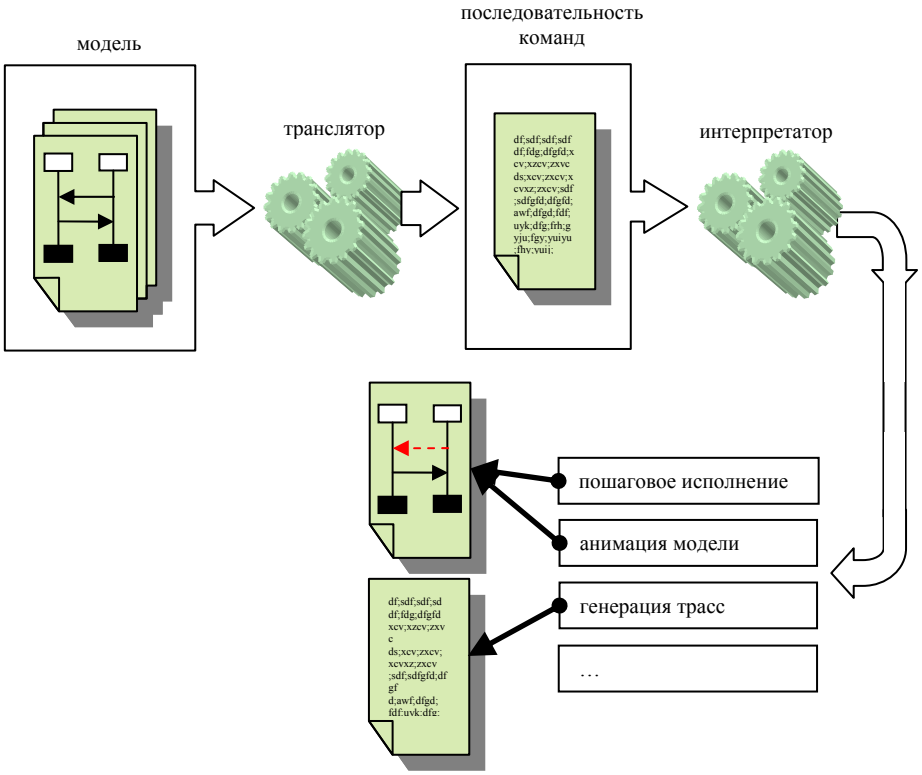

Рис. 3. Использование виртуальной машины.

На Рис. 3 схематически представлен процесс исполнения модели при помощи виртуальной машины. Модель подаётся на вход транслятору, который переводит её в набор команд виртуальной машины. Далее интерпретатор осуществляет исполнение команд в заданном пользователем режиме. Кроме того, в процессе исполнения может происходить запись отладочной информации.

При визуализации исполнения здесь, как и в случае использования автоматов, необходимо определять соответствие между конструкциями, употребляемыми в моделях, и их образами в последовательности команд виртуальной машины.

Исследуя рисунки 1, 2 и 3, можно заметить, что и непосредственная интерпретация, и использование автоматов являются частными случаями метода исполнения моделей, основанного на виртуальной машине. Действительно, в первом случае этап трансляции является вырожденным, и внешнее представление модели одновременно образует последовательность команд для интерпретационной компоненты виртуальной машины. Не представляет большой сложности определить набор команд для представления автоматов, в таком случае этап трансляции модели в такую последовательность команд представляет собой ни что иное, как генерацию автоматов по модели. Таким образом, последний из представленных способов исполнения моделей является, очевидно, наиболее гибким.

5. Создание виртуальной машины

При создании виртуальной машины для некоторого класса задач необходимо выполнить следующие этапы:

1. Выявить весь объём данных, которые в процессе выполнения задачи могут подвергаться преобразованию. Эти данные образуют поле зрения виртуальной машины.
2. Определить набор команд, который, подобно командам универсальной ЭВМ, будет служить для записи решения конкретной задачи. Этот набор является специфичным для данного класса задач. Последовательность команд из этого набора, определяющая решение конкретной задачи, записывается в программное поле виртуальной машины и не меняется в процессе её функционирования.
3. Реализовать интерпретационный компонент виртуальной машины, представляющий собой совокупность механизмов. Посредством этих механизмов виртуальная машина осуществляет интерпретацию команд, находящихся в программном поле.
4. Реализовать транслятор из внешнего представления модели в последовательность команд виртуальной машины.

Для виртуальной машины должен быть определён способ её использования, то есть:

- способ передачи входной информации и формирования начального поля зрения;
- способ активизации (запуска) системы;
- способ останова системы;

- способ извлечения переработанной информации.

Тот факт, что при создании виртуальной машины заданию подлежит набор команд, определяет большее удобство её применения. Для использования автоматов должны быть разработаны методы сведения моделей к автоматам, а решение данной задачи может представлять значительную сложность. В случае же виртуальной машины мы сами имеем возможность взять такой набор команд, который наилучшим образом будет отражать специфику выбранных для исполнения моделей. При этом сам процесс перевода модели в последовательность подобранных таким образом команд не потребует серьёзных усилий. Конечно же, процесс подбора команд должен представлять собой компромисс между простотой перевода моделей в команды и легкостью собственно интерпретации составленных из таких команд последовательностей.

5.1. Комплексирование работы нескольких виртуальных машин

5.1.1. Виртуальная машина как строительный компонент

Взглянем ещё раз на то, что представляет собой виртуальная машина. Она состоит из некоторого интерпретатора, исполняющего последовательность известных ему команд и имеющего доступ к области памяти для хранения данных. Рассмотрим, что получится, если допустить возможность существования в пределах одного сеанса исполнения нескольких таких интерпретаторов. Поскольку программное поле виртуальной машины остаётся неизменным в течение всего процесса её функционирования, то несколько интерпретаторов могут работать с одной и той же последовательностью команд или каждый иметь свою собственную. Возможна также реализация, в которой различные интерпретаторы будут предназначены для работы с разными наборами команд. Таким образом, задача исполнения модели может быть разделена на подзадачи, для каждой из которых будет создан отдельный набор команд и его интерпретатор со своей моделью памяти. Кроме того, возможен также случай совместной работы нескольких интерпретаторов над общей областью памяти для хранения данных, но, поскольку данные в процессе исполнения модели могут изменяться, такой подход сопряжён с дополнительными трудностями и его использование не всегда оправдано.

Таким образом, набор команд в совокупности с его интерпретатором и моделью памяти образуют своеобразный строительный компонент. Из таких компонентов при создании системы исполнения моделей можно составлять различные комбинации. Остаётся только вопрос о способах комплексирования работы таких компонентов, наибольший интерес из которых представляют два, описанные ниже.

5.1.2. Вызов виртуальной машины

Пожалуй, одним из наиболее полезных способов комплексирования виртуальных машин является вызов одной виртуальной машины из другой. При его реализации вызываемые и вызывающие машины могут как совпадать, так и существенно различаться по своей структуре. В наборе команд той

виртуальной машины, из которой происходит вызов, выделяется команда или команды для порождения и вызова другой виртуальной машины. Порождённая машина получает ссылки на своё поле данных и последовательность команд, после чего запускается. Поведение вызывающей машины при этом может варьироваться: либо ожидание завершения вызванной машины, либо продолжение своего выполнения без ожидания завершения вызванной. Следует отметить, что вызовы могут быть многократно вложенными.

Для иллюстрации вышесказанного рассмотрим две возможных реализации вызова виртуальных машин.

1. Виртуальные машины различны, имеют разные наборы команд и различную структуру области для хранения данных. Вызвавшая машина после осуществления вызова блокируется до окончания работы вызванной, после чего продолжает своё выполнение. Такой способ вызова полезен, например, при исполнении моделей на языке MSC 2000 [3], который имеет два уровня: HMSC (high-level MSC) и собственно MSC (message sequence charts — диаграммы последовательности сообщений). HMSC-диаграмма представляет собой граф потока управления, в узлах которого помещены ссылки на MSC-диаграммы. Таким образом, виртуальная HMSC-машина будет являться вызывающей, а MSC-машина — вызываемой.
2. Виртуальные машины имеют одинаковую структуру, работают с одним набором команд, но имеют отдельные экземпляры области данных. Вызвавшая машина не блокируется после вызова, а продолжает своё выполнение. Такая организация полезна для обработки альтернатив, которые часто встречаются во многих моделях и призваны отражать различные варианты поведения системы. Допустим, что в процессе своей работы виртуальная машина A_1 дошла до такого места, где ей нужно обработать n альтернатив. В этом случае A_1 создаёт $n-1$ своих копий A_2, A_3, … A_n. При этом копированию подвергается и поле зрения, поскольку предполагается, что до разветвления поведение всех машин было одинаковым. Таким образом, всего приходится по одной машине на каждую альтернативу, и каждая из машин продолжает своё выполнение в пределах назначенной альтернативы. Следует отметить, что с этих пор все машины действуют абсолютно независимо и ничего не знают друг о друге, хотя они и могут разделять один и тот же набор команд. Такой способ может быть применён для обработки альтернатив в языке MSC [3] на HMSC диаграммах и альтернативных выражений на самих MSC диаграммах. Данный пример поднимает ещё один важный вопрос об организации параллельной работы нескольких виртуальных машин.

5.1.3. Параллельное исполнение

При комплексировании работы виртуальных машин могут возникать ситуации, когда нужно организовать их параллельное выполнение. Это может потребоваться, например, в случае необходимости рассмотрения всех

альтернатив сразу. Кроме этого, некоторые языки моделирования имеют в своём составе специальные конструкции для обозначения того, что элементы модели должны работать параллельно.

Для организации параллельной работы нескольких виртуальных машин можно использовать механизм потоков (thread), предоставляемый многозадачными операционными системами. В этом случае каждая виртуальная машина запускается в отдельном потоке.

Если же существует необходимость организации более детального контроля над параллельной работой виртуальных машин, то можно вручную организовать их псевдопараллельное выполнение. В этом случае в качестве единицы работы отдельной виртуальной машины можно использовать её команды. Каждая из параллельно работающих машин поочерёдно получает право сделать один шаг, после чего она возвращает управление. Для реализации этого можно использовать менеджер, содержащий список всех параллельно работающих машин и раздающий им право сделать очередной шаг на основании выбранной стратегии.

Если в системе реализованы вызовы одной виртуальной машины из другой, то при получении машиной права сделать очередной шаг возможны два варианта:

1. Не существует машины, вызванной из данной машины. В этом случае машина выполняет одну команду из своего программного поля и возвращает управление.
2. Из данной машины A вызвана машина B, и машина A ожидает её завершения. В таком случае машина A делегирует право сделать очередной шаг машине B. Машина B выполняет шаг и возвращает управление в машину A, после чего та тоже возвращает управление.

От того, насколько эффективно будет организована работа с моделями, напрямую зависит успех MDD. Один из наиболее удобных и эффективных путей познания пролегает через эксперимент. Именно поэтому исполнение моделей способно значительно улучшить и ускорить их понимание. В данной статье был предложен способ исполнения моделей, основанный на использовании виртуальных машин. Показаны его основные преимущества по сравнению с непосредственной интерпретацией и конечными автоматами, которые широко применяются в данной области. При проектировании системы исполнения моделей, основанной на виртуальной машине, разработчик обладает большей свободой в принятии решений, поскольку команды виртуальной машины создаются исходя из потребностей выбранной предметной области. Этим же обусловлена и простота перевода модели в последовательность таких команд. Кроме того, виртуальные машины обладают широкими возможностями комплексирования.

Литература:

1. *Брукс Ф.* Мифический человеко-месяц или как создаются программные системы. — Пер. с англ. — СПб.:Символ-Плюс, 2001. — 304 с.: ил.

2. *Harel D.* Biting the silver bullet: Toward a brighter future for system development // Computer. — 1992. — Jan. — P. 8-20.
3. ITU-T Recommendation Z.120 Message Sequence Charts (MSC).
4. *Ladkin P. B., Leue S.* What do Message Sequence Charts mean? // Formal Description Techniques VI, IFIP Transactions: Proc. of the 6th Intern. Conf. on Formal Description Techniques / Ed. by Tenney R. L., Amer P. D., Uyar M. U. — Boston, 1994. P. 301-316.
5. *Selic B.* The Pragmatics of Model-Driven Development // IEEE Software. — 2003. — Vol.20, No.5. — P. 19-25.