УДК 681.3.06

ГРАФИЧЕСКОЕ ПРОГРАММИРОВАНИЕ В УПРАВЛЕНИИ ДИНАМИКОЙ МОДЕЛИРОВАНИЯ СЛОЖНЫХ ПРОЦЕССОВ

М.М. Ластовченко

Международный научно-учебный центр информационных технологий и систем НАН Украины
03680, Киев 187, проспект Академика Глушкова, 40.
Тел.: 526 1175, e-mail: zobr@ua.fm

Рассмотрены основные требования пользователя к графической программной среде, которая управляет моделированием сложных систем. Также анализируются методы реализации UML алгоритмов такой среды.

In the present work the main user requirements to the graphic program environment are viewed, which manager simulation complicated systems. Also methods of UML algorithms realization of such environment are analyzed.

Введение

Главной и наиболее существенной модификацией языков графического программирования сегодня является обеспечение управления моделированием с помощью метамоделей управления развитием (МУР, MDD – model-driven development) [1–3]. Эти программные средства должны обеспечивать и эскизное (первый этап), и техно-рабочее проектирование (последующие этапы). Особенно это важно для разработки распределенных аппаратно-программных средств для сложных территориально-распределенных систем телекоммуникации и транспорта [2, 3].

Целый ряд производителей разработали основанные на последней версии языка UML (UML-2.0) инструментально-технологические комплексы, которые поддерживают значительно более высокий уровень интеллектуализации проектирования, чем традиционные CASE-средства (computer-aided software engineering) [4–6]. Еще больших успехов удалось достичь при модификации языка описаний и спецификаций SDL-2000, модули которого могут быть интегрированы в модели с завершающей спецификацией на нижних уровнях моделирования, где определяются требования к компонентам системы и процессам их функционирования [7–9].

Для поддержки более высокого уровня интелектуализации проектирования необходимо ввести в язык UML более точные способы спецификации процессов, чем это было в стандарте UML1, который был, главным образом, создан как вспомогательное средство для неформального формирования и отображения общих целей (общих моделей) проектирования [1].

Сегодня введен ряд достаточно интеллектуальных методов структурирования языка UML и расширены за счет этого возможности графического моделирования [3, 10]. Появилась теория метамоделирования с преобразованием цепочки моделей, которые определяют конкретные требования к тому, как должен быть создан (специфицирован) сам язык моделирования [5, 6, 10]. Однако, все еще отсутствует системотехническая теория создания такого языка графического моделирования, который был бы сравним с языками, использующими современную теорию проектирования средств программирования.

Таким образом, можно определить требования к модификации одного из базовых языков графического програмирования, а именно модификации последующих версий UML 2.0.

1. Увеличить степень точности определении компонент языка за счет модификации его семантической среды, что обеспечит поддержку спецификации на более высоких системотехнических уровнях МУР.

2. Создать новую организацию модульной структуры языка в рамках новой развитой архитектуры взаимодействия модулей, которые не только сделают язык более доступным но и будут содействовать интеграции его в инструментальные средства спецификации.

3. Ввести интерфейсы обеспечивающие интеграцию модулей языка UML-2 с модулями языка спецификаций SDL-2000 на нижних уровнях описания процессов функционирования.

Исходя из вышеуказанных требований, в работе анализируется каждая из указанных проблем дальнейшей модификации языка как языка управления моделированием сложных процессов функционирования. (Наиболее типовыми сложными процессами могут быть процессы функционирования мультисервисных сетей реального времени (МСС РВ)).

1. Повышение точности описаний в графических диаграммах за счет модификации семантической среды языка моделирования

Существующие сегодня концепции моделирования выражаются с использованием недостаточно точных на базе естественных языков описаний, что не дает возможности вести корректную спецификацию процессов. Это

сейчас считается достаточным, поскольку большинство языков моделирования используется либо для документирования, либо для создания эскиза дизайна (design sketching). Задача здесь состоит в передаче основных свойств дизайна, оставляя работу с деталями спецификации на время последующих этапов реализации проекта, что сводит на нет взаимодействие различных разработчиков: системотехников и программистов [10].

В отличие от большинства языков моделирования уже в первой версии (UML1) предпринята попытка специфицировать первое стандартизованное определение UML, использующее MDD [3-6]. Метамодель была определена с помощью элементарного подмножества UML и дополнена набором формальных ограничений, написанных на языке конструкций объекта (ЯКО, OCL – Object Constraint Language) [3]. Это подмножество UML, которое первоначально охватывало концепции определенные в графических диаграммах классов UML, названо обобщенной объектно-ориентированной средой (ООС, MOF – Meta-Object Facility). Это подмножество как среда выбрано потому, что его можно было использовать для определения других языков моделирования. Оно представляло формальную спецификацию абстрактного синтаксиса UML, называемого так из-за его независимости от реальной системы обозначений, или конкретного синтаксиса, т. е. текста и графики, который использовался для представления моделей.

Таким образом, эта среда моделирования определила набор правил, которые можно было использовать для проверки грамматической правильности данной модели. Например, такие правила позволяли определять, что подключение к двум UML-классам при изменении состояния конечного автомата является некорректным. Однако степень точности, используемая в этой первой UML-метамодели, была недостаточной для поддержки всего потенциала MDD. В частности, спецификация семантик (или значений) UML-концепций моделирования оставалась неадекватной для таких основанных на MDD операциях как автоматическое генерирование кода или формальная верификация.

В новой версии UML 2.0 степень точности моделей была значительно увеличена по отношению к UML за счет введения следующих средств [4–6]:

1. Существенным рефакторингом инфраструктуры метамодели. Инфраструктура языка UML 2.0 стала включать в себя набор многоуровневых концепций моделирования и шаблонов, которые в большинстве случаев слишком абстрактны для того, чтобы быть использованными непосредственно в моделировании процессов поддерживаемых программными приложениями. Однако их относительная простота дает им возможность быть значительно более точными по своей семантике, что лучше соответствует правилам грамматической корректности. Такие корректно настроенные концепции затем комбинируются различными способами для создания более сложных концепций моделирования на высшем пользовательском уровне.

2. Расширенным и более точным описанием семантики. Определение семантики концепций моделирования в UML 1 было во многом проблематичным. Многоуровневое описание было крайне неравномерным, некоторые уровни имели расширенные и детализированные описания (например, конечные автоматы), при этом как другие имели краткое объяснение, либо не имели их вовсе. В UML 2.0 определена значительно лучшая семантика языка, особенно в ключевой области – основах динамики поведения процессов функционирования.

3. Четким определением динамики семантической среды. Спецификация UML 2.0 определяет большую часть критических задач воспроизведения процессов, которые в семантике версии UML 1 не решались.

Поскольку самым эффективным средством повышения степени точности моделей является последнее средство, постольку рассмотрим его более подробно. Сущность формирования семантической среды языка UML 2.0 можно проиллюстрировать на примере моделирования сложных процессов функционирования мультисервисной сети реального времени МСС РВ. Рассматривая вначале кибернетическую среду реальных процессов (функционирования) МСС РВ. На рис. 1 показана функциональная структура кибернетической среды составляющих объектов и процессов функционирования МСС РВ.

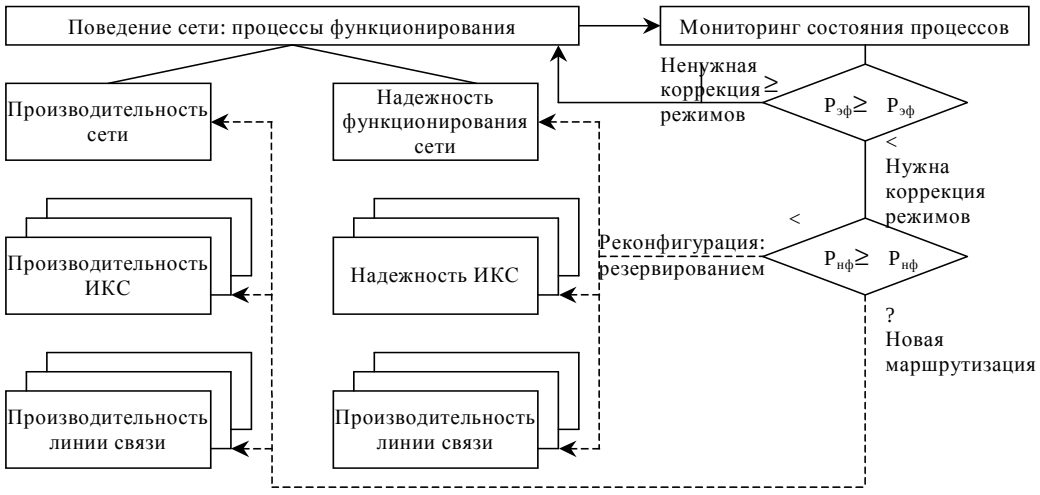

Рис. 1. Функциональная структура кибернетической среды мультисервисной сети реального времени

На рис. 2 показана новая (модифицированная) структура семантической среды языка UML 2.0, которая должна обеспечивать воспроизведение динамики процессов показаных на рис. 1.

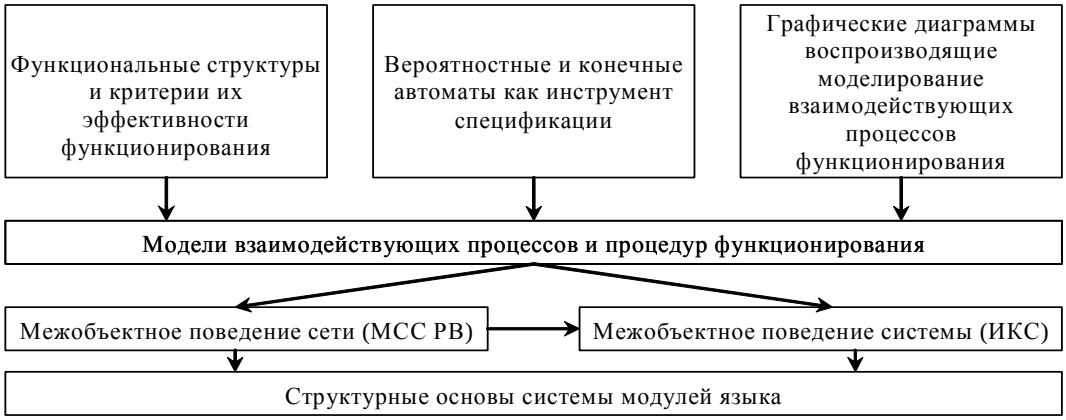

Рис. 2. Семантическая среда модифицируемого языка UML-2

В этой среде четко определены следующие аспекты формирования диаграмм графического моделирования.

1. Структурная семантика ссылок и экземпляров во время исполнения моделирования.
2. Взаимоотношения между структурой и воспроизводимым поведением процессов функционирования.
3. Основы семантики, которые заложены в модули, обеспечивающий воспроизведение сложных процессов.

2. Создание модульной структуры языка моделирования

Другим аспектом в повышении степени точности графических диаграмм в UML 2.0, является то, что определение свойств языка, стало точнее и объемнее даже без учета новых функциональных возможностей. Это, в свою очередь, обеспечило создание новой организации модульной структуры языка.

Исходя из того, что язык UML предназначен для решения сложных проблем современного графического программного обеспечения, а также из того, чтобы упростить проблему сложности языка, UML 2.0 был разбит на модули.

Разбиение выполнено таким образом, чтобы можно было разрешить избирательное использование модулей языка, как для разработки наиболее сложного сетевого программного обеспечения (СПО) МСС РВ, так и для разработки распределенных аппартно-програмных средств для МСС РВ (таблица) [3–5].

Таблица. Языковые модули UML 2.0

Языковый модуль	Предназначение
Действия (Actions) (Процедуры процессов функционирования, обеспечивающий поддержание надежности и передачи)	Исходная функциональная структура как основа моделирования с указанием на выполнение заданных процедур
Активности (Activities) (Режимы функционирования (работоспособности передачи) с анализом надежности и производительности)	Моделирование динамики процессов: сбоев и передачи потока данных и сигналов управления
Классы (Classes) (Иерархия и структура классов составляющих (блоков) темы сети)	Функциональная структура как основа моделирования базовых составляющих структур
Компоненты (Components) (Структура взаимозависимых компонент системы)	Сложное моделирование с иерархией компонент структуры
Развертывания (Deployments) (Режимы процедуры)	Моделирование процесса развертывания

Окончание табл.

Общие поведения(General Behaviors) (Сценарии состояний и переходов между ними)	Функциональная структура системы как основа (общая база семантики поведения) моделировании во времени
Информационные потоки (Information Flows)	Моделирование потока абстрактных данных
Взаимодействия (Interactions)	Моделирование процессов взаимодействия объектов
Модели (Models)	Организация модели
Профайлы (Profiles)	Настройка языка
Конечные автоматы (State Machines)	Моделирование управляемого событиями процесса
Вероятностные автоматы (Probabilities Machines)	Моделирование управляемого случайными событиями процесса
Структуры (Structures)	Моделирование комплексной иерархической структуры
Шаблоны (Templates)	Моделирование шаблонов
Прецеденты (Use Cases)	Моделирование процессов неформального поведения

Для эффективного взаимодействия модулей при их интеграции необходимо было разработать многоуровневую архитектуру языка.

3. Создание архитектуры языка моделирования

Существующие сегодня концепции моделирования выражаются с использованием недостаточно точных на базе естественных языков описаний, что не дает возможности вести корректную спецификацию процессов. Это сейчас считается достаточным, поскольку большинство языков моделирования используется либо для документирования, либо для создания эскиза дизайна (design sketching). Задача здесь состоит в передаче основных свойств дизайна, оставляя работу с деталями спецификации на время последующих этапов реализации проекта, что сводит на нет взаимодействие различных разработчиков: системотехников и программистов [10].

Общий вид архитектуры языка показан на рис. 3. Архитектура как основа построения структуры языка, охватывает все общие концепции (такие как классы и ассоциации), включающие набор вертикальных подъязыков или языковых модулей. Каждый из них предназначен для моделирования конкретной формы структуры системы или аспекта ее поведения (таблица). Эти вертикальные языковые модули, в общем, независимы друг от друга, поэтому можно использовать их независимо друг от друга. (Это отличается от применения UML 1, где формализм "активность" полностью основывался на формализме "конечный автомат").

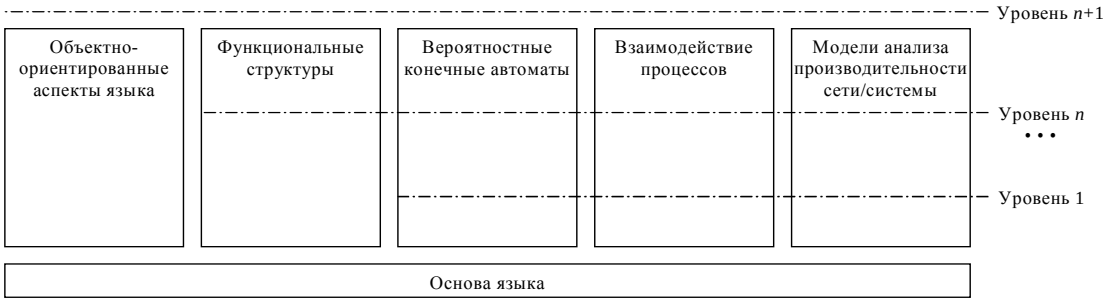

Рис. 3. Архитектура языка UML 2.0

Вертикальные языковые модули иерархически организованы в три уровня, каждый последующий уровень добавляет новые функциональные возможности моделирования к уже доступным на нижнем уровне. Это придает дополнительную размерность модульности, обеспечивая внутри каждого языкового модуля возможность использования только определенных подмножеств.

Такая архитектура дает возможность использовать при моделировании только то подмножество UML, которое больше подходит для создаваемой диаграммы высшего уровня.

В UML 2.0 определены только три типа уровней совместимости модулей (1, *n* и *n*+1, рис. 3), которые соответствуют уже упоминавшимся уровням иерархии языковых модулей. Они определены таким образом, чтобы модели уровня (n) являлись бы совместимыми с моделями любого старшего уровня (*n*+1 и т.д.). Точнее, инструментальное средство, которое совместимо с данным уровнем, должно импортировать модули модели (без потерь информации) из инструментальных средств, совместимых с любым уровнем, равным или меньшим его уровню.

Необходимо также отметить, UML 2.0 формально определяется четырмя уровнями (Level 0), но это внутренний уровень, предназначенный для использования только разработчиками самых инструментальных средств графического программирования. Для этого определены подуровня совместимости:

- совместимость абстрактного синтаксиса;
- совместимость конкретного синтаксиса (то есть, системы обозначений UML);
- совместимость абстрактного и конкретного синтаксиса;
- совместимость абстрактного и конкретного синтаксиса, и совместимость со стандартом обмена диаграммами.

Таким образом, существует максимум 12 различных комбинаций совместимости с четкими взаимосвязями между ними (например, совместимость абстрактного и конкретного синтаксиса согласуется только с совместимостью конкретного синтаксиса или только абстрактного синтаксиса). Следовательно, в моделях UML 2.0 обмен модулями между совместимыми инструментальными средствами различных производителей становится не только теоретической, но и главное практической возможностью.

Все расширения архитектуры выполнены с использованием одинакового подхода: рекурсивное применение одного и того же базового набора концепций на различных уровнях абстракции. Это означает, что уже можно скомбинировать элементы модели данного типа в модули, которые, в свою очередь, можно будет использовать как строительные блоки для комбинирования таким же способом на следующем уровне абстракции, и т.д. Это аналогично процедурам в языках программирования, которые могут быть вложены внутрь других процедур с любой желаемой глубиной.

Таким способом расширены следующие функциональные возможности моделирования: 1. синтез сложных структур; 2. анализ производительности; 3. анализ взаимодействия процессов; 4. расширение аппарата конечных автоматов и повышение спецификации требований с помощью языка.

На первые три возможности приходится более 90 % новых возможностей в модифицированном UML 2.0 [3–6].

4. Пример динамики моделирования сложных процессов

Существующие сегодня концепции моделирования выражаются с использованием недостаточно точных на базе естественных языков описаний, что не дает возможности вести корректную спецификацию процессов. Это сейчас считается достаточным, поскольку большинство языков моделирования используется либо для документирования, либо для создания эскиза дизайна (design sketching). Задача здесь состоит в передаче основных свойств дизайна, оставляя работу с деталями спецификации на время последующих этапов реализации проекта, что сводит на нет взаимодействие различных разработчиков: системотехников и программистов [10].

Сущность динамики моделирования заключается в развитии диаграмм обобщенной спецификации до диаграмм детальной спецификации. В первой версии UML 1 отсутствовали два важных свойства развития диаграмм [6–8]:

1. Способность к повторному использованию последовательностей, которые могли повторяться в контексте более высокого уровня последовательностей. Например, последовательность, подтверждающая пароль, могла появляться в нескольких контекстах. Без возможности выделить такую повторяющуюся последовательность в отдельных модулях ее приходилось определять множество раз.

2. Способность адекватно моделировать различные сложные потоки сигналов управления, которые являлись обычными при представлении взаимодействий процессов функционирования сложных систем.

Вместе с тем для сложных процессов проблема определения и обозначения сложных взаимодействий всесторонне проработана в области телекоммуникаций. Они же использованы как основа для представления взаимодействий в UML 2.0. В UML 2.0 представлено взаимодействие как отдельного именованного модуля моделирования, так и последовательность межобъектных взаимодействий произвольной сложности. Кроме того, такое взаимодействие может быть параметризованным для спецификации шаблонов контекстно-независимого взаимодействия. Более того, взаимодействия могут служить в качестве операндов в сложных конструкциях управления, таких как циклы (например, если данное взаимодействие может понадобиться повторить несколько раз) или ветвления.

В работах Б. Селика [3–5] приведен пример использования языка UML 2.0, который определяет несколько полезных конструкций моделирования процессов сложного типа. Этот подход как инструмент для моделирования сложного сквозного поведения можно применять на любом уровне декомпозиции.

На рис. 4 показана диаграмма модели банковской системы с расширенным взаимодействием [4, 5].

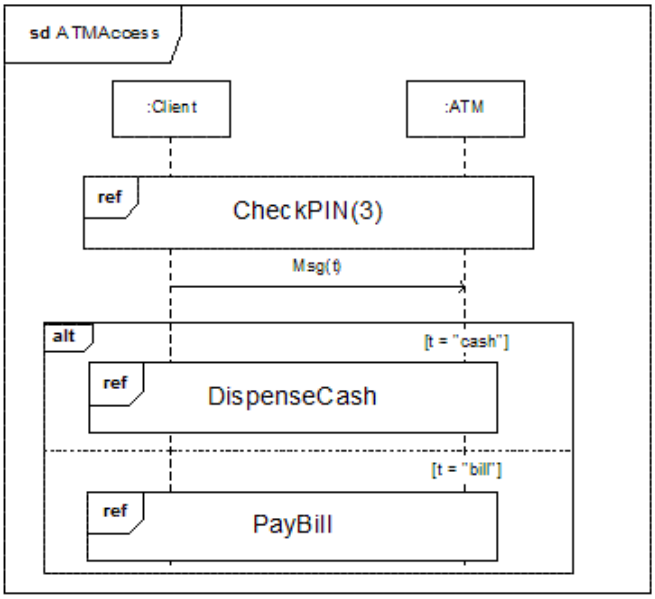

Рис. 4. Диаграмма модели UML в виде архитектуры сложного взаимодействия отображающего обобщенную спецификацию

В данном случае взаимодействие ATM Access сначала «инициирует» другую транзакцию меньшего уровня с именем Check PIN. Необходимо обратить внимание, что последнее взаимодействие имеет «сигнальный» параметр (в данном случае, максимальное количество попыток ввода PIN-кода перед аннулированием транзакции). После этого клиент посылает асинхронное сообщение, указывающее требуемый тип взаимодействия и (в зависимости от указанного значения) выполняется либо взаимодействие Dispense Cash, либо взаимодействие Pay Bill.

В семантике действий UML основной элемент модели — это процедура, которая содержит несколько действий (подпроцедур). В большинстве языков программирования этим действиям соответствует последовательность операторов, но в UML последовательное исполнение не считается непреложным фактом. Процедура всегда имеет контекст. Например, процедура может представлять тело операции класса или преобразование состояния, а семантика определяет, что происходит, когда эти действия (процедуры) выполняются. Семантика действий в этом случае, умышленно, не включает в себя корректную нотацию, которую предлагает SDL [7, 8].

Машины состояний UML не уделяют преобразованиям того внимания, которого те заслуживают, что проявляется в отсутствии четкой нотации для их действий [3, 4]. Поскольку диаграммы состояний UML, в общем и целом, ориентированы на состояние, они «игнорируют» то, что все это происходит во времени. В SDL ситуация практически полностью противоположная – представление диаграмм SDL ориентировано на преобразования [9, 10]. Здесь процедуры переходов отображают действия. Таким образом, эти два представления в значительной степени дополняют друг друга: ориентация UML на состояние дает возможность получить общее описание структуры представления объекта (системы), а ориентация на преобразование в диаграммах SDL позволяет конкретизировать процессы в деталях, сохраняя контекст диаграммы UML. Более того, введение семантики MSC/SDL позволяет получить необходимые временные диаграммы с показателями взаимодействия [9, 11].

Описываемое поведение процесса состоит из операторов (процедур), в виде заключенных в своего рода скобки сигналов, которые несут в себе информацию о действиях: процессах и их процедурах и имеют конкретное семантическое значение (рис. 5) [4, 5].

Диаграмма модели, детализированной спецификации (рис. 5) является последующим развитием диаграммы модели обобщенной спецификации (рис. 4) с раскрытием сложного процесса взаимодействия. Символ ввода указывает сигнал или операцию, которые инициируют преобразование и могут содержать список параметров; символ задачи обычно указывает на присваивание; символ вывода показывает сообщение, которое агент посылает определенному объекту; символ сохранения определяет сигналы, которые необходимо отслеживать; «звездочка» в символе сохранения служит указанием на сигналы и операции, для которых состояние точно не определено, и показывает, что они должны быть обработаны в последующих состояниях. Символ замечания содержит декларации атрибутов (переменных) и ему предшествует ключевое слово DCL, что позволяет отличить замечание от обычных комментариев.

Таким образом, особое внимание в спецификации проектируемых сложных процессов должно уделятся процедурам, как части динамик преобразуемых процессов. Однако, для процедур, которые являются, в свою очередь, частью операций, принципы остаются теми же. На рис. 5 показаны действия SDL на основе семантики действий UML [12, 13].

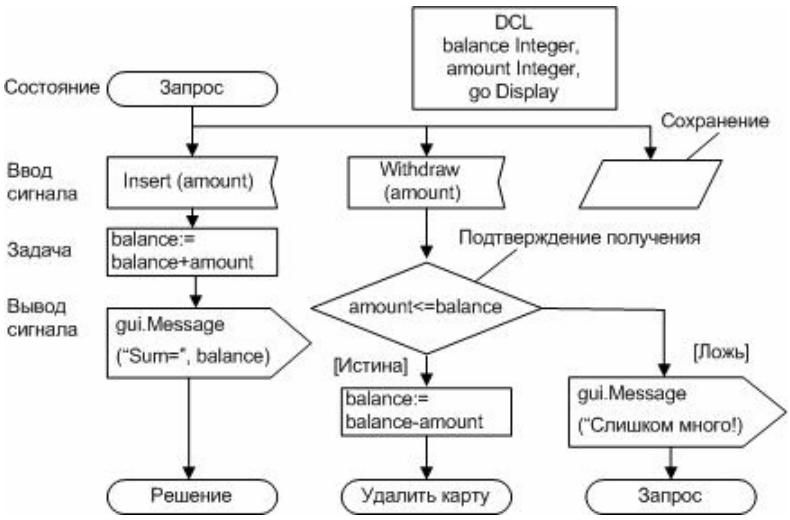

Рис. 5. Диаграмма модели UML в виде архитектуры преобразований
определяющих детализированную спецификацию взаимодействия процедур процесса

В заключение можно сделать вывод о том, что реализация предлагаемых концептуальных положений создания и введения графического программирования в управление моделированием при проектировании и разработке сложных систем (транспортных или телекоммуникационных сетей) требует интеграции существенно модифицированных языков UML и SDL.

Потенциал, который скрыт за этой мощной комбинацией формальных моделей (диаграмм) UML и SDL, воспроизведения конкретных процессов функционирования сложных систем, приведет к появлению новых технологий графического программирования и соответствующих методов разработки графических диаграмм относящихся к так называемым метамоделями управления развитием (МУР, MDD – model-driven development) [3–5]. При этом определяющей особенностью MDD должно быть то, что модели должны становиться основными артефактами графических образов программы перемещая, таким образом, акцент с соответствующего кода программы к графическим образам процессов. В этих условиях инженер-системотехник получит принципиально новый инструментально-технологический комплекс интелектуального проектирования [2, 9, 12].

1. *Гомма Х.* UML: Проектирование систем реального времени, распределенных и параллельных приложений. – М.: "ДМК", 2002 – 698 с.
2. *Ионин Г.Л.* Описание моделирования систем с использованием SDL // Рига – «АВТ», 1982. – 1. – С. 30–34.
3. *Selic B, Rumbauyn I.* "Using UML of Modeling Complex Real Time Systems" // Unpublished white paper. – 1998. – 4. – 28 p.
4. *Selic B.* On the Semantic Foundations of Standard UML 2.0, in Bernardo, M., and Corradini, F. (eds.), Formal Methods for the Design of Real-Time Systems, Lecture Notes in Computer Scince. – Springer-Verlag, 2004. – Vol. 3185 – 48 p.
5. *Selic, B.* Unified Modeling Lanuage Version 2.0 // Unpublished white paper. – 2005. – 28 p.
6. *Ларман К.* Применение UML как шаблонов проектирования. – М.: «Вильямс», 2004. – 624 с.
7. *ITU-T* Rec. Z-100 «The Specification and Deseription Language (SDL)». – Geneva; 2000. – 191 p.
8. *Reed B.* SDL-2000 for New Millennium Systems. – Telelektronic; 2002 – P. 81–96.
9. *Ластовченко М.М., Макаренко М.Н., Марущак В.И.* Интеллектуализация программных средств описания и спецификации телекоммуникационных систем и процессов их функционирования // Проблемы программирования. – 2006. – 4. – С. 55–67.
10. *Ластовченко М.М., Биляк В.И., Горбунов И.Э., Русецкий В.Е.* Концепция формирования программной среды описания и моделирования для спецификации требований к проектам беспроводных сетей // Проблемы программирования. – 2006. – № 2/3. – С. 177–187.
11. *ITU-T* Rec Z 120 Message Sequence Chart (MSC). – Geneva IX 99; 2000. – 42 p.
12. *Bjorkander M.* Craphical Programming using UML and SDL // IEEE Computer. – 2000. – 2. – P. 30–35.
13. *ITU-T* Rec. Z-109 «SDL Combinet with UML» // http://www.itu.int/itu-doc/itu-t/tec/z/index.htm.