УДК 004.4'242

М.В. Галочкин, П.И. Соснин

СРЕДСТВА ПСЕВДОКОДОВОЙ ПРОГРАММИРУЕМОЙ ГРАФИКИ В ПРОЕКТИРОВАНИИ АВТОМАТИЗИРОВАННЫХ СИСТЕМ

Галочкин Михаил Владимирович, *окончил факультет информационных систем и технологий УлГТУ, магистр техники и технологии по направлению «Информатика и вычислительная техника», аспирант по направлению 05.13.12 «Системы автоматизации проектирования (по отраслям)». Имеет статьи в области САПР. [e-mail: m.galochkin@ulstu.ru].*

Соснин Петр Иванович, *заслуженный работник высшей школы РФ, доктор технических наук, профессор, окончил радиотехнический факультет Ульяновского политехнического института. Заведующий кафедрой «Вычислительная техника» УлГТУ. Имеет многочисленные труды в области концептуального проектирования автоматизированных систем. [e-mail: sosnin@ulstu.ru].*

Аннотация

В статье представляются средства псевдокодовой программируемой графики поддержки проектирования автоматизированных систем на концептуальном этапе. Специфика такого подхода характеризуется отображением «block and line» диаграмм в семантическую память вопросно-ответного типа. Структура ячеек памяти и возможность управления атрибутами ячеек и их значениями позволяют приписать элементам диаграмм семантику, с которой возможны программируемые действия. Рисунок конкретной диаграммы можно преобразовать в псевдокодовое программное описание, с которым можно производить полезные манипуляции как с исходным программным кодом. Кроме того, псевдокодовое представление диаграммы можно использовать для ее визуализации в поле специализированного графического редактора. Более того, отображение диаграммы на ее псевдокодовое представление открывает возможность для полезных преобразований диаграмм, что способствует включению в процесс проектирования механизмов управления моделями. Предлагаемые средства реализованы и проверены в инструментально-моделирующей среде WIQA (Working In Questions and Answers).

Ключевые слова: автоматизированное проектирование, графовые трансформации, диаграммы, концептуальное проектирование, псевдокодовое программирование, проектное управление моделями.

PSEUDOCODE PROGRAMMABLE GRAPHICS TOOLS IN AUTOMATED SYSTEMS DESIGNING

Mikhail Vladimirovich Galochkin, *graduated from the Faculty of Information Systems and Technologies of Ulyanovsk State Technical University; Master of Engineering and Technology in Informatics and Computer Engineering; Post-Graduate Student with a specialization in Design Automation Systems; an author of articles in the field of CAD. e-mail: m.galochkin@ulstu.ru.*

Petr Ivanovich Sosnin, *Honored Worker of the Higher School of the Russian Federation, Doctor of Engineering, Professor; graduated from the Radio Engineering Faculty of Ulyanovsk Polytechnic Institute; Head of the Computer Science Department at Ulyanovsk State Technical University; an author of numerous works in the field of conceptual design of computer-aided systems. e-mail: sosnin@ulstu.ru.*

Abstract

The article presents the tools for the pseudocode programmable graphics designed for support of automated system engineering at a conceptual stage. The particularity of the approach offered is defined by mapping «block and line» diagrams into a semantic memory of question-answer type. The structure of memory cells and capability of control of cell attributes and values ascribe the semantics, with which programmable actions are liable to occur, to elements of diagrams. The specific diagram image can be converted to a pseudocode program description that can be modified the same way as a source programme code. In addition, the pseudocode diagram presentation can be used for its visualization in the special graphics editor field. Furthermore, mapping the diagram onto its pseudocode presentation offers the opportunity for useful conversions of diagrams that contributes to add model control mechanisms to the design process. The decisions offered have been implemented and tested in the tool-and-modeling environment WIQA (Working In Questions and Answers).

Key words: computer-aided design, graph transformations, diagrams, conceptual design, pseudocode programming, design model control.

Введение

В инструментально-технологическом сопровождении процессов проектирования автоматизированных систем широко используются средства текстового и табличного сопровождения, а также средства чертежной и псевдографики, обслуживающие создание диаграмм и виде «block-and-line» схем. Эффективные и мощные средства графики предоставляются для моделирования реальных объектов и процессов, а также для их образных представлений. Проектировщикам предоставляется и эффективная схемотехническая графика, которая в основном ориентирована на чертежную регистрацию результатов проектирования [1]. Такие представления и преобразования не только поддерживают работу на этапах принятия проектных решений, но и доведены до их нормативного включения в технологические и производственные процессы.

Использование графических моделей в современной программной инженерии стало не только повсеместным, но и вошло в ряд стандартов (UML, BPMN, BPEL и т. д.). Акцент в разработках сложных автоматизированных систем в последнее время существенным образом сместился в область концептуального проектирования, в процессе которого создаются и используются множества концептуальных моделей с разной степенью общности и детализации. Роль таких моделей выполняют «block and line» схемы разных типов, чаще называемые «диаграммами». Эволюция «block and line» средств моделирования сложных систем, интенсивно использующих программное обеспечение, привела к разработке унифицированного языка моделирования (Unified Modeling Language, UML).

Отметим, что текстовое и табличное сопровождение в САПР заимствует практически все достигнутое в разработках текстовых и табличных процессоров, но чаще всего только в привязках к решению задач формирования и регистрации проектной документации. Однако как в поддержке работ с графикой, так и в работе с текстами практически отсутствует выход на автоматизированное решение задач семантики, занимающих очень важное место в оперативной работе проектировщика и особенно процессах принятия проектных решений.

Для проектировщика выход на ту семантику, для которой полезно ее графическое представление, начинается с первых шагов его взаимодействия с техническим заданием и особенно принципиален на этапах концептуального проектирования, когда только еще нащупываются потенциальные прототипы решений, когда выход на реальную геометрию и нормативную графическую (образную) регистрацию преждевременен. На этом этапе основная информация поступает к проектировщику через тексты, во взаимодействии с которыми он практически лишен автоматизированной графической поддержки. Это одна из причин, которые послужили основанием для разработки средств понятийно-образной поддержки, раскрываемых ниже.

Вторая причина связана с активным внедрением в практику проектирования автоматизированных систем средств разработки, управляемых моделями (model driven development), в основу которых положены не только графические модели, но и их автоматизированные и автоматические трансформации, доводящие там, где это удается, до исходных и/или исполняемых программных кодов [2].

Сущность предлагаемых решений заключается в следующем: взаимодействия с текстами практически полезно дополнить понятийно-образной поддержкой, в основе которой лежит построение «block and line» диаграмм с программно доступными составляющими. Кроме того, на уровне программного доступа к составляющим диаграмм открыт доступ и к их атрибутике, выражающей семантику блоков (blocks) и связей между ними (lines).

Именно поэтому проектировщику, решающему определенную проектную задачу, предоставляется возможность оперативно создавать полезные ему диаграммы и нагружать их составляющие семантикой, которая соответствует сложившейся ситуации. Кроме того, проектировщику предоставляется возможность программируемой трансформации диаграмм, нацеленной на их использование в решении задач проектирования, управляемых моделями. Предлагаемые решения реализованы в вопросно-ответной моделирующей среде WIQA.

1 Анализ существующих подходов

Существующие подходы к анализу «block and line» моделей с целью перевода последних в код (для исполнения или отладки) можно разделить на несколько видов в зависимости от того, какие артефакты создаются после перевода моделей:

1. Перевод моделей в исполняемый код на ЯВУ. К средствам, реализующим данный подход, можно отнести Altova UModel. Данный инструмент позволяет переводить 14 видов диаграмм UML в исполняемый код на ЯВУ (java, c# …) [3]. Такие средства чаще всего «заточены» под конкретную графическую нотацию (например, UML) и языки программирования (C#, Java …) и не обладают возможностью расширяться.

2. Перевод моделей в запросы к данным (scientific workflow system). Такой подход реализован в Tavena и позволяет манипулировать с данными посредством удаленных сервисов [4]. Код в данном случае представляет собой очередь запросов к различным сервисам и запросы на изменения данных. Не подразумевает использование человека в качестве «интеллектуального процессора» [5].

3. Перевод моделей в очередь задач. Такой подход характерен для Oracle BPEL Process Manager и других workflow систем. В данном случае модели трансформируются в инструкции, которые должны выполнять либо компьютер, либо человек (Human Tasks). Часто поддерживается расширение нотации за счет ввода дополнительных объектов (написания кода на ЯВУ, описывающее поведения объекта).

Разрабатываемый подход отличается от представленных, хотя он наиболее близок к третьему виду. В отличие от первого, в нем подразумевается отсутствие привязки к конкретному языку и графической нотации, что реализуется путем возможности «быстрой» расширяемости поддерживаемых блоков (за счет добавления новых правил трансформации) и использованием псевдокода (псевдокод в отличие от ЯВУ наиболее близок к естественному языку). В отличие от 2 подхода, разрабатываемый подход делает акцент на задачах, выполняемых человеком, не концентрируется на данных, а оперирует более высокоуровневыми абстракциями, характерными для этапа концептуального проектирования (такими как задача, исполнитель и т. д.). Последний подход не подразумевает использование прецедентов, тогда как разрабатываемый подход переводит диаграммы в вопросно-ответную память, что автоматически подразумевает использование прецедентов и сохранение наиболее удачных решений в качестве паттернов.

2 Требования к средству анализа «block and line» диаграмм

В связи с анализом существующих решений и были сформированы требования к разрабатываемому подходу и средству на базе инструментального комплекса WIQA, реализующего этот подход.

1. Отсутствие привязки к конкретному ЯВУ, так как это накладывает жесткие ограничения на синтаксис языка проектирования, что является нецелесообразным на концептуальном этапе. Необходимо использовать язык, наиболее приближенный к естественному, который обладает большой выразительной мощью.

2. Отсутствие привязки к конкретной графической нотации. Гибкость в расширение поддерживаемых графических нотаций и семантических блоков. Это следствие вытекает из первого и подразумевает, что средство будет легко расширяемым.

3. Возможность работы с базой опыта и паттернов. Лучшие решения должны сохраняться в вопросно-ответной памяти.

4. Возможность экспериментирования на моделях (перевод в код, исполнение, отладка). Как было сказано выше, модель сама по себе несет мало пользы, если нет средства, позволяющего исполнять и отлаживать такую модель.

Обобщенная схема средства анализа приведена на рисунке 1.

Описанные выше ограничения требуют отсутствия привязки к определенным нотациям. Это подразумевает, что в общем случае любая «block and line» диаграмма может быть проанализирована и переведена в псевдокод для дальнейшей обработки. Но после проведения ряда экспериментов пришлось ограничиться диаграммами, где связи имеют направления. Это стало необходимо, так как любая

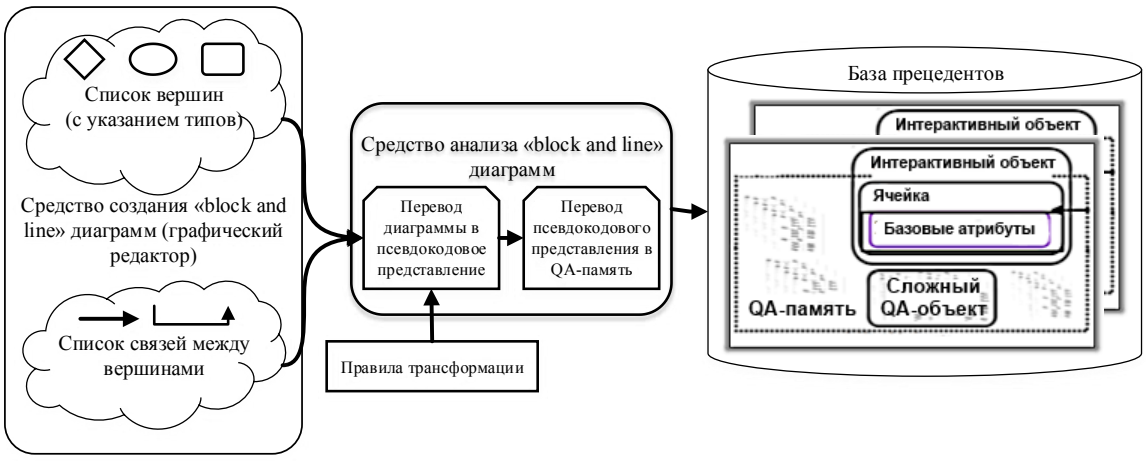

Рис. 1. Обобщенная схема средства анализа «block and line» диаграмм

модель описывает движение некой сущности по различным этапам, требует указания направления (в качестве таких сущностей выступают данные, управление, задачи и т. д.), то есть средство позволяет анализировать любые «block and line» диаграммы, в которых связи имеют направления.

3 Атрибуты вершин, связей, правил трансформации и диаграмм

«Block and line» диаграммы в общем случае явно не содержат атрибутики, кроме взаимного расположения элементов (координаты). Но данной информации недостаточно для правильной интерпретации семантических блоков, из которых состоит диаграмма. Под семантическими блоками подразумеваются группы вершин и связей, объединенных одной смысловой нагрузкой. Например, в рамках диаграмм активностей UML 2.x можно выделить следующие семантические блоки: условия, циклы (с пред- и постусловием), блоки распараллеливания/слияния, последовательности активностей (2 и более активности) и т. д. Проанализировав существующие подходы, стало очевидным, что минимальный набор атрибутов должен включать тип вершины (в большинстве нотаций определяется той фигурой, которая представляет вершину), а также имя вершины. Для удобства оперирования вершинами каждая должна содержать уникальный идентификатор, хотя это не является необходимым условием.

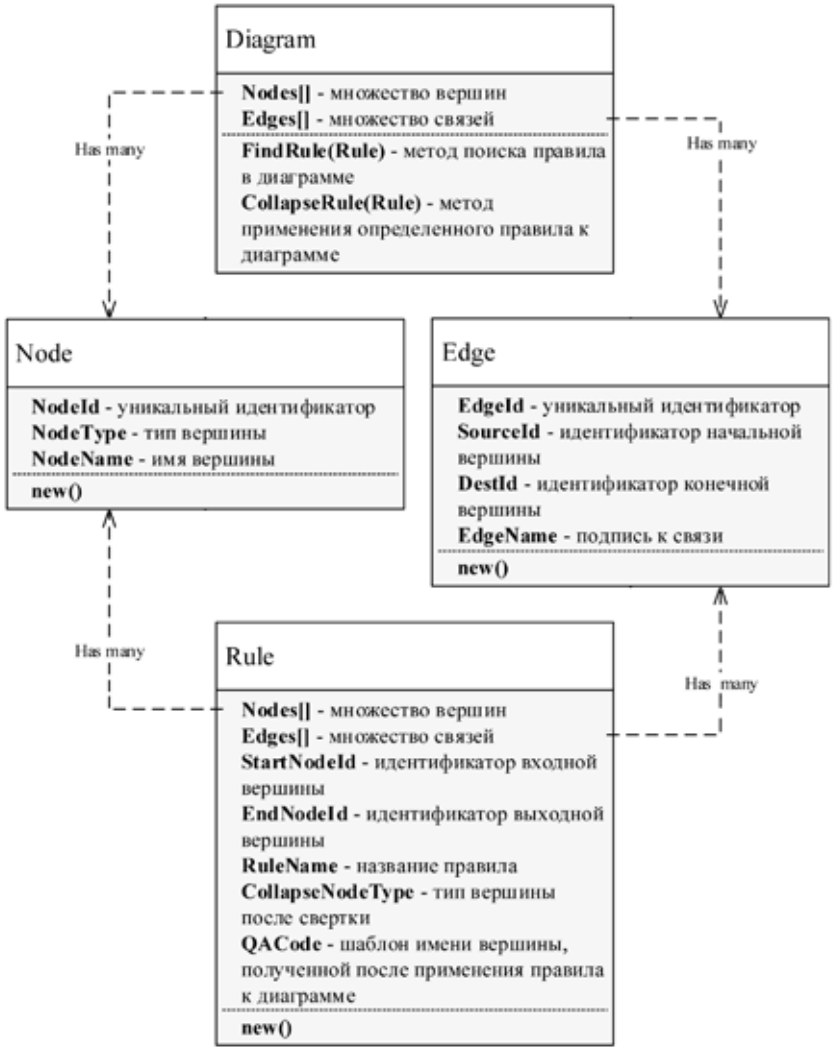

Рис. 2. Диаграмма классов для вершин, связей и правил трансформации

Связи содержат атрибуты начального элемента (идентификатор) и конечного. Дополнительным атрибутом будет являться значение связи или как во многих нотациях – подпись к связи [3]. Это информация является необходимой, так как позволяет выделять такие блоки, как условия (во многих нотациях само условие является атрибутом связи).

Правила трансформации представляют собой граф (более подробно смотри раздел 5), состоящий из массива вершин и связей, а также имеющий начальную и конечную вершины (рис. 2).

Реальная диаграмма классов, использующаяся при разработке данного подхода, отличается наличием служебных полей в классах, а также вспомогательными методами, которые были исключены из данного представления для большей наглядности. Часть исходных кодов доступны по адресу: https://github.com/mixail7/modeler.

4 Графовые трансформации

В последнее десятилетие, особенно в западной литературе, набирает популярность подход, известный как разработка, управляемая моделями (Model Driven Development) [6, 7]. Одним из основополагающих понятий подхода являются «графовые трансформации» [8, 9]. Правила для графовых трансформаций представляют собой пары графов, имеющих обозначения Left Hand Side(LHS) и Right Hand Side (RHS) [8]. При применении правила на граф G происходит поиск всех подграфов, соответствующих LHS, и замена найденных по соответствующему правилу на RHS. Формальное описание графовых трансформаций, используемых в нашем подходе для анализа «block and line» диаграмм, приведено ниже:

$G := (V, E)$, V – множество вершин, $E := (V_i, V_j)$, (1)

где $V_i, V_j \in G$,

$Rule := (G_1, G_2)$,

где $G_1 := (V_1, E_1, V_{н1}, V_{к1}), V_{н1}, V_{к1\ j} \in V_1$,

$G_2 := (V_2, E_2), E_2 = \varnothing$, (2)

если $V_1 \subseteq V \wedge E_1 \subseteq E$, то (3)

$G' := (V', E')$, где $V' = V \cup V_2 \setminus V_1$, (4)

$E' = E \cup E'_{н2} \cup E'_{к2} \setminus E_1 \setminus E'_{н1} \setminus E'_{к1}$,

где G' – граф, полученный после применения правила Rule к графу G,

$E'_{н2}$ – множество связей, исходящих из любой вершины графа G и входящих в V_2 ($V_{н1}$),

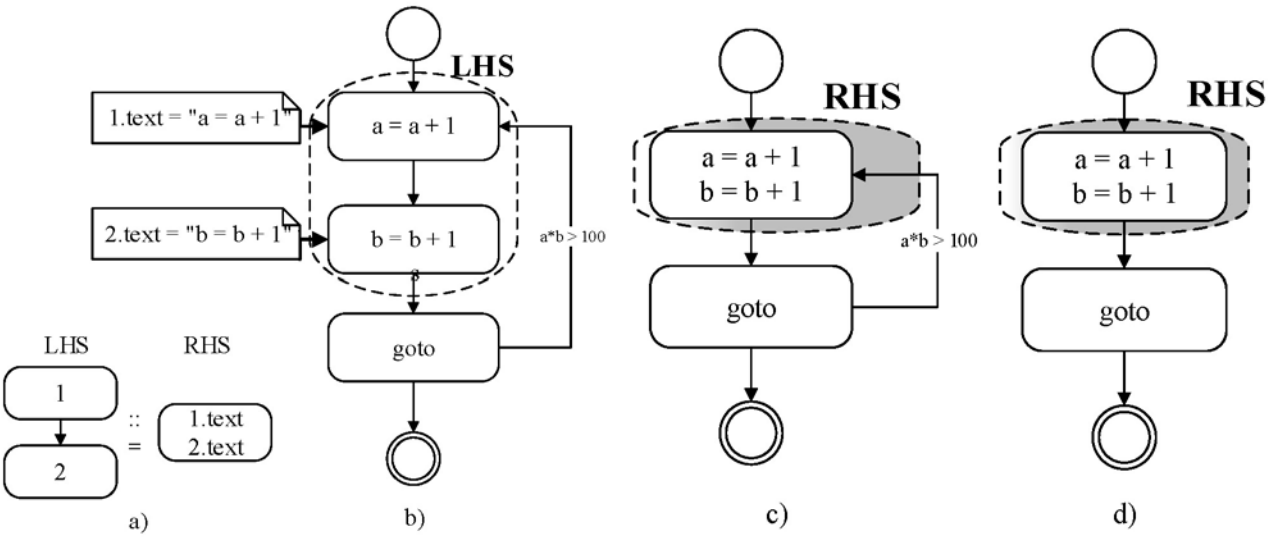

Рис. 3. Пример применения графовой трансформации (a) на граф (b) с сохранением связей (c) и без сохранения (d)

$E'_{к2}$ – множество связей, исходящих из V_2 ($V_{к1}$) и входящих в любую вершину графа G,

$E'_{н1}$ – множество связей, исходящих из любой вершины графа G и входящих в $V_{н1}$,

$E'_{к1}$ – множество связей, исходящих из $V_{к1}$ и входящих в любую вершину графа G.

В общем случаем графовые трансформации подразумевают поиск подграфа и замену на другой подграф. В нашем случае, как видно из описания, граф для замены является вырожденным (множество связей является пустым множеством). Это необходимо для того, чтобы на каждом этапе получать более простой граф и, в конечном итоге, путем ряда сверток получить граф, состоящий из одной вершины.

Графовая трансформация может быть неоднозначной, если существуют связи, которые ведут к вершинам подграфа (LHS). Данное противоречие представлено на рисунке 3 на примере простой диаграммы активностей.

Такое противоречие решается несколькими способами [10]. В нашем подходе мы исключаем возможность применения трансформации, если внутренние вершины LHS графа содержат связи с вершинами, не входящими с LHS. Для этого в формальное описание трансформации были введены $V_{н1}$, $V_{к1}$ – соответственно начальная и конечная вершины для правила. Под начальной вершиной понимается та, которая может иметь входящие связи из любых вершин графа, не входящих в LHS, а под конечной – та, которая может иметь исходящие связи в любые вершины графа, не входящие в LHS.

Для того чтобы не потерять информацию о блоке, к которому применяется свертка, в вершине, полученной, после применения трансформации, сохраняется определенный «код». Правило для получения кода представлено в вершине RHS на рисунке 3a, а код, полученный после применения правила, - на рисунке 3c и 3d (RHS).

5 Представление правил трансформации

Как было отмечено выше, правила трансформации представляют собой пары графов LHS и RHS. На практике правила трансформации содержат дополнительную информацию, необходимую для сохранения данных о семантическом блоке после его свертки. Такой информацией является «код», полученный из QACode (см. рис. 2).

Название тегов xml на рисунке 4 схожи с полями, представленными на диаграмме классов (рис. 2). Типами

```
1  <rule name="2 action">
2      <nodes>
3          <node id="1">ActyvityDiagram_Pal.AD_Action_Element</node>
4          <node id="2">ActyvityDiagram_Pal.AD_Action_Element</node>
5      </nodes>
6      <startNodeId>1</startNodeId>
7      <endNodeId>2</endNodeId>
8      <edges>
9          <edge id="3" sourceId="1" destId="2" name="" />
10     </edge>
11     <QACode>1.text\r\n2.text</QACode>
12     <collapseNodeType>ActyvityDiagram_Pal.AD_Action_Element</collapseNodeType>
13 </rule>
```

Рис. 4. Xml-представление правила трансформации, показанного на рисунке 3a

вершин в данном случае являются названия элементов в палитре редактора «block and line» диаграмм (входит в средство анализа).

6 Вопросно-ответное представление модели. Использование прецедентов

При создании моделей перед ними ставится ряд задач, которые они призваны решать. Модели несут в себе решение, т. е. содержат в себе ответы на поставленные вопросы [11]. В QA-моделях вопросы присутствуют явно в форме «объектов-вопросов» и неявно. Инструментальная среда WIQA имеет богатый функционал для манипуляций такими моделями, поэтому, разработав средство перевода диаграмм в QA-модели, мы получаем ряд возможностей. Детальное описание паттернов использования среды WIQA выходит за рамки статьи и подробно представлено в работах профессора Соснина П.И. [5, 11–15]. Следует остановиться на одном из важных аспектов – использовании базы прецедентов, который выделялся как преимущество перед другими подобными подходами (раздел 1). Процесс решения задач на концептуальном этапе формально представлен на рисунке 5.

На первом этапе после возникновения проблемы и формирования задач происходит обращение к базе опыта с целью нахождения похожих задач, которые были успешно решены ранее. В англоязычной литературе такой этап принято называть «retrieve», а подход называется рассуждениями по прецедентам (case-based reasoning). В случае, если схожая задача решалась ранее, на следующем этапе произойдет адаптация найденного решения под новые требования. На этом этапе происходит «отладка» полученного решения (с помощью перевода в псевдокод). Заключительным этапом выступает сохранение прецедента в базу опыта посредством псевдокодовой QA-модели. Расширяя базу опыта, мы автоматически повышаем вероятность нахождения схожих моделей.

Эффективность данного метода (case-based reasoning) сильно зависит от предметной области, в которой он используется. Задачи, которые необходимо решать, должны иметь тенденцию к повторению и обладать принципом регулярности (подобные задачи должны иметь подобные решения). В противном случае вероятность нахождения схожих решений в базе опыта будет стремиться к нулю. По мнению экспертов, большинство задач в области концептуального проектирования обладает таким свойством [12, 10, 16], и в настоящий момент ведутся работы в данном направлении с целью создания «общей базы прецедентов» [16].

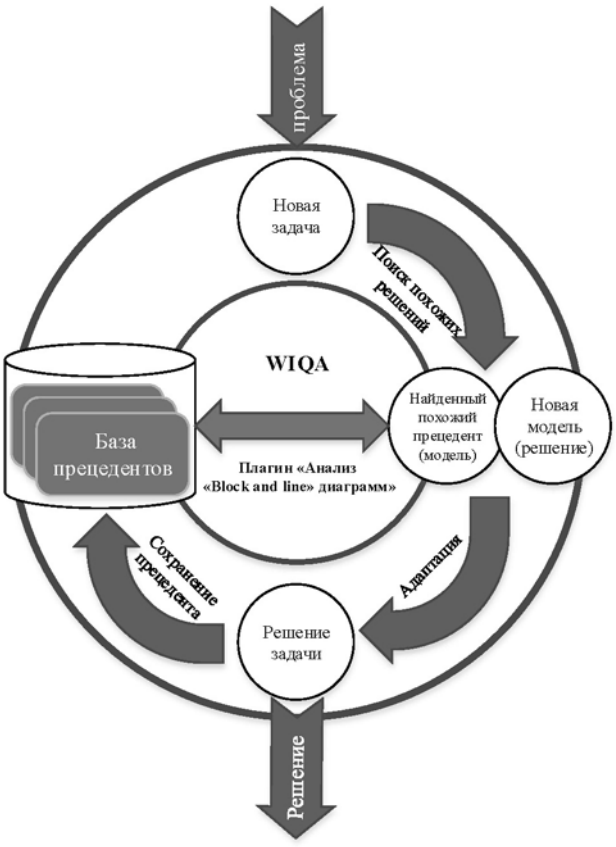

Рис. 5. Процесс решения задач в WIQA

Заключение

В статье представлен подход к автоматизации процесса построения и исполнения (тестирования) графических моделей на концептуальном этапе проектирования. Существующие решения не позволяют оперировать с базой опыта (прецедентов) либо используют определенные нотации (перевод на ЯВУ), которые в значительной степени сужают возможности разработчика. Как отмечалось выше, концептуальное проектирование в большей степени является творческим процессом, и такие ограничения являются недопустимыми.

Разработанное и описанное в статье средство позволяет автоматизировать процесс построения «block and line» моделей, выполнять их перевод, исполнение и отладку (пошаговое исполнение) на языке, приближенном к естественному (псевдокод). Автоматическое сохранение переведенных моделей в вопросно-ответную память открывает большие возможности к повторному использованию проектных решений, доказавших свою эффективность.

СПИСОК ЛИТЕРАТУРЫ

1. Tufte E. R. The Visual Display of Quantitative Information. – Graphics Press, 2001.

2. Ponirakis A. L. Improving Software Efficiency: Automated Program Transformation for Reclaiming Execution Efficiency // Navysbir. – 2012. – URL: http://www.navysbir.com/n13_1/N131-061.htm.

3. UML Tool. – URL: http://www.altova.com/umodel.html (дата обращения: 9.01.2015).

4. Why use work flows? – URL: http://www.taverna.org.uk/introduction/why-use-workflows/ (дата обращения: 7.01.2015).

5. Sosnin P. Role «Intellectual Processor» in Conceptual Designing of Software Intensive Systems, ICCSA'2013 // The 11th International Conference on Computational Science and Applications, Part III, LNCS 7973 Springer, Heidelberg, 2013, pp. 1–16.

6. France R., Rumpe B. Model-Driven Development of Complex Software: A Research Roadmap. // Proc. FOSE '07 2007 Future of Software Engineering. Washington, 2007, pp. 37–54.

7. Beydeda S., Book M., Gruhn V. (Eds.). Model-Driven Software Development. // Springer-Verlag, Heidelberg, 2005, 464 p.

8. Lars Grunske, Leif Geiger, Albert Zündorf. Using Graph Transformation for Practical Model-Driven Software Engineering. // Model-Driven Software Development, Heidelberg, Germany, Springer, 2005, pp. 91–118.

9. Beydeda S., Book M., Gruhn V. Model-Driven Software Development. – Berlin : Springer-Verlag, 2005. – 464 p.

10. Haan Johan. 15 reasons why you should start using Model Driven Development // theenterprisearchitect. – 2009. – URL: http://www.theenterprisearchitect.eu/archive/2009/11/25/15-reasons-why-you-should-start-using-model-driven-development (дата обращения: 7.01.2015).

11. Соснин П.И. Архитектурное моделирование автоматизированных систем. – Ульяновск : УлГТУ, 2007. – 146 с.

12. Соснин П.И. Концептуальное моделирование компьютеризованных систем – Ульяновск : УлГТУ, 2008. – 198 с.

13. Соснин П.И. Вопросно-ответное программирование человеко-компьютерной деятельности – Ульяновск : УлГТУ, 2010. – 240 с.

14. Соснин П.И. Онтологическая поддержка концептуального экспериментирования в вопросно-ответных моделирующих средах // Тр. Конгресса по интеллектуальным системам и информационным технологиям : науч. издание в 4-х т. – М. : Физматлит, 2014. – Т. 1. – С. 488–495.

15. Pseudo-Code Simulation of Designer Activity in Conceptual Designing of Software Intensive Systems // Proceedings of the 27th European Conference on Modeling and Simulation, ECMS 2013, 2013. pp. 85–92.

16. Methods Need Theory, By Ivar Jacobson and Bertrand Meyer, August 06, 2009. – URL: http://www.drdobbs.com/architecture-and-design/methods-need-theory/219100242 (дата обращения: 9.01.2015).

REFERENCES

1. Tufte E. R. *The Visual Display of Quantitative Information*. Graphics Press, 2001.

2. Ponirakis A. L. Improving Software Efficiency: Automated Program Transformation for Reclaiming Execution Efficiency. *Navysbir.*, 2012. Available at: http://www.navysbir.com/n13_1/N131-061.htm.

3. UML Tool. Available at: http://www.altova.com/umodel.html (accessed 09.01.2015).

4. Why use workflows? Available at: http://www.taverna.org.uk/introduction/why-use-workflows/ (accessed 7.01.2015).

5. Sosnin P. Role 'Intellectual Processor' in Conceptual Designing of Software Intensive Systems. *The 11th International Conference on Computational Science and Applications (ICCSA'2013),* Part III, LNCS 7973 Springer, Heidelberg, 2013, pp. 1–16.

6. France R., Rumpe B. Model-Driven Development of Complex Software: A Research Roadmap. *Proc. Future of Software Engineering (FOSE '07 2007),* Washington, 2007, pp. 37–54.

7. Beydeda S., Book M., Gruhn V. (Eds.). *Model-Driven Software Development*. Springer-Verlag, Heidelberg, 2005, 464 p.

8. Lars Grunske, Leif Geiger, Albert Zündorf. Using Graph Transformation for Practical Model-Driven Software Engineering. *Model-driven Software Development,* Heidelberg, Germany, Springer, 2005, pp. 91–118.

9. Beydeda S., Book M., Gruhn V. *Model-Driven Software Development*. Berlin, Springer-Verlag, 2005. 464 p.

10. Haan Johan. 15 reasons why you should start using Model Driven Development. *theenterprisearchitect,* 2009. Available at: http://www.theenterprisearchitect.eu/archive/2009/11/25/15-reasons-why-you-should-start-using-model-driven-development (accessed 07.01.2015).

11. Sosnin P.I. *Arkhitekturnoe modelirovanie avtomatizirovannykh system* [Architectural Modeling of Automated Systems]. Ulyanovsk, UlSTU Publ., 2007. 146 p.

12. Sosnin P.I. *Kontseptualnoe modelirovanie kompiuterizovannykh system* [Conceptual Modeling of Computer-Aided Systems]. Ulyanovsk, UlSTU Publ., 2008. 198 p.

13. Sosnin P.I. *Voprosno-otvetnoe programmirovanie cheloveko-kompiuternoi deiatelnosti* [Question-Answer Programming the Human-Computer Interaction]. Ulyanovsk, UlSTU, Publ., 2010. 240 p.

14. Sosnin P.I. Ontologicheskaia podderzhka kontseptualnogo eksperimentirovaniia v voprosno-otvetnykh modeliruiushchikh sredakh [Ontological Approval of Conceptual Experimenting in the Question-Answer Modeling Environment]. *Tr. Kongressa po intellektualnym sistemam i informatsionnym tekhnologiiam : nauch. izdanie v 4-kh t.* [Proc. of the Congress on Intelligence Systems and Information Technologies. Scientific Publication in 4 Volumes], Moscow, Fizmatlit Publ., 2014, vol. 1, pp. 488–495.

15. Pseudo-Code Simulation of Designer Activity in Conceptual Designing of Software Intensive Systems. *Proc. of the 27th European Conference on Modeling and Simulation, ECMS 2013*, 2013. pp. 85–92.

16. Methods Need Theory, By Ivar Jacobson and Bertrand Meyer, August 06, 2009. Available at: http://www.drdobbs.com/architecture-and-design/methods-need-theory/219100242 (accessed 09.01.2015).