ИНФОРМАТИКА И ПРОГРАММИРОВАНИЕ

УДК 004.652.8
DOI: 10.17223/19988605/41/6

А.М. Бабанов, А.В. Петров

РЕАЛИЗАЦИЯ РЕПОЗИТОРИЯ ERM-МОДЕЛИ В CASE-СИСТЕМЕ ORACLE DESIGNER

Описаны основные принципы и архитектурные особенности реализации прототипа CASE-репозитория, созданного для апробации постулатов и идей ERM-модели. В ходе анализа базовых средств в качестве платформы для создания репозитория ERM-схем была взята CASE-система Oracle Designer. Этот выбор позволил в сжатые сроки разработать исследовательский прототип репозитория, позволяющий развернуть тестирование и отладку реальной работы многих других подсистем исследовательского CASE-инструмента для ERM-модели.
Ключевые слова: ERM-модель; репозиторий; CASE; Oracle Designer.

Развитию инструментальных средств, методов и методик разработки информационных систем (ИС) всегда уделялось и уделяется большое внимание. В конце прошлого столетия появились и активно использовались преимущественно при создании систем баз данных (БД) так называемые CASE-системы (CASE – Computer Aided System Engineering) – программные инструменты автоматизации процесса разработки ИС. Методы системного анализа и проектирования, использующиеся в таких инструментах, требуют высокой математической и мировоззренческой культуры, а также определенных самодисциплины и пунктуальности. Это обстоятельство, наряду с большой стоимостью таких средств, привело к тому, что широкого распространения они не получили. Что касается системных методов анализа и проектирования, то их упрощенные варианты с успехом используются при «ручной» разработке.

Однако исследователи методов анализа и проектирования не прекратили своих усилий в решении задачи автоматизировать труд разработчиков ИС [1–4]. Они идут по пути увеличения отдачи CASE-систем в плане повышения степени автоматизации, производительности труда, а также качества результата [1, 5]. Для достижения поставленных перед этими исследованиями целей необходимо совершенствовать используемые модели анализа и проектирования, а также инструменты межмодельного преобразования и окончательной генерации исполняемых артефактов. Тестирование и отладка вновь предлагаемых средств невозможны без создания исследовательского прототипа будущей CASE-системы, что само по себе представляет достаточно сложную задачу. Настоящая статья посвящена разработке ядра такой системы – репозитория. Отличительной особенностью статьи является ориентация на использование для информационного моделирования семантической модели данных «Сущность – Связь – Отображение» [2].

1. Назначение репозитория в CASE-системах и требования к разрабатываемому репозиторию

Webster's Dictionary определяет термин «репозиторий» как «предмет или человек, воспринимаемый в качестве центра аккумуляции информации или хранилища» [6. С. 181]. «Еще в недавней истории разработки программного обеспечения (ПО) роль репозитория мог играть лишь человек – программист, который должен был помнить расположение всей информации, относящейся к проекту разработки ПО, знать еще нигде не записанную информацию и восстанавливать потерянную информацию. Использование человека в качестве “центра аккумуляции информации и хранилища” не давало хороших результатов. Сегодня репозиторий является предметом – базой данных, которая служит и центром аккумуляции информации, и хранилищем информации о разработке ПО. Ролью человека (разработчика ПО) теперь

является взаимодействие с репозиторием посредством CASE-средства, интегрированного с этим репозиторием» [6. С. 182].

Репозиторий CASE-системы – это множество инструментов и структур данных, с помощью которых достигают интеграции данных и диалоговых средств, а также данных различных инструментов между собой. Он обеспечивает очевидные функции БД, но дополнительно репозиторий выполняет или способствует выполнению следующих функций [6]:

1) поддержка целостности данных, которая в случае репозитория представляет собой обеспечение непротиворечивости схем ИС;

2) распределение информации между несколькими разработчиками, специализирующимися на создании различных артефактов проекта;

3) взаимодействие данных репозитория и диалоговых инструментов;

4) взаимодействие данных репозитория, созданных в различных инструментах;

5) поддержка определенной методики анализа и проектирования;

6) стандартизация документации (графической и текстовой);

7) хранение и управление данными сложной структуры в условиях нормализованной схемы БД;

8) обеспечение высокоуровневого (большей частью – графического) интерфейса с пользователями;

9) управление процессом разработки;

10) трассировка требований на объекты репозитория;

11) поддержка версионности данных репозитория;

12) поиск зависимостей в данных и управление изменениями;

13) аудит изменений.

Конечно, исследовательский прототип репозитория не обязан обеспечивать выполнение всех указанных функций, большая часть которых связана с поддержкой непосредственной работы по проектам. Однако если многие из этих функций будут реализованы в базовых инструментах разработки, это будет полезно.

Наряду с указанными функциональными требованиями у исследовательского прототипа репозитория есть ряд важных нефункциональных требований:

– система должна иметь программный интерфейс, позволяющий работать со схемами ИС посредством внешних приложений. Таковыми будут являться графический редактор ERM-схем, генератор реляционных схем и другие инструменты, которые в последующем будут реализованы;

– система должна иметь универсальный пользовательский интерфейс, предоставляющий диалоговый доступ ко всем объектам репозитория. Информацию о текущем состоянии репозитория, объектах, ассоциациях, свойствах хранимых элементов зачастую удобнее получать посредством некоторого встроенного диалогового инструмента;

– поскольку ERM-модель (для поддержки которой в первую очередь создается репозиторий) активно развивается, ее метасхема будет часто изменяться. Следовательно, архитектура прототипа должна быть гибкой и предоставлять возможность повторного использования ранее разработанных компонентов.

2. Варианты создания ERM-репозитория

Поскольку разрабатываемая система ориентирована на хранение и предоставление доступа к аналитической и проектной информации, а также должна осуществлять взаимодействие многочисленных разработчиков и других приложений, можно выделить два альтернативных способа ее реализации. Первый способ заключается в создании собственного репозитория «с нуля» на основе долговременного хранилища данных. В качестве хранилища данных может выступать, например, реляционная СУБД. Второй способ заключается в расширении какого-либо уже имеющегося CASE-средства.

В первом случае предстоит решать все задачи репозитория, кроме управления данными. Да и последнее придется осуществлять не в терминах метамодели репозитория, и уж тем более не в терминах ERM-модели, а с использованием понятий, структур и языков, поддерживаемых базовой (как правило, реляционной) СУБД. Пожалуй, единственным неоспоримым преимуществом этого способа является

потенциальная возможность создания максимально эффективного (по расходам памяти и скорости реакции) репозитория.

Второй подход предполагает наличие некоторого готового репозитория, в схеме которого уже, как правило, реализованы метасхемы ER-модели и реляционной модели. Последняя необходима в любом CASE-репозитории, поскольку именно реляционная модель поддерживается большинством СУБД и поэтому является целевой для исполняемой схемы БД. Кроме того, в этом репозитории уже могут быть решены многие (если не все) универсальные задачи, упомянутые ранее и стоящие перед любым репозиторием, какую бы семантическую модель данных он ни поддерживал.

Второй вариант является более предпочтительным для исследовательского прототипа, поскольку многие функции, необходимые репозиторию, уже имеются в базовом инструменте, остается только реализовать в нем поддержку дополнительной метасхемы ERM-модели. Действительно, если существует такой инструмент, который при небольших доработках удовлетворит поставленным ранее требованиям, процесс разработки будет значительно сокращен. А проигрыш в эффективности, неизбежно сопутствующий такому решению, не играет существенной роли для исследовательского прототипа.

3. Описание репозитория Oracle Designer

В качестве базовой платформы для создания репозитория ERM-схем была взята CASE-система Oracle Designer (OD) [7]. Центральной частью этой системы является репозиторий, содержащий спецификации проекта на всех его этапах и обеспечивающий согласованную работу всех его участников, какие бы роли они ни играли в проекте. Для доступа к репозиторию и управления им кроме специализированных диалоговых инструментов имеется универсальное средство – навигатор по объектам репозитория (Repository Object Navigator – RON), позволяющий просматривать и модифицировать практически все объекты, хранящиеся в репозитории [7].

Основным достоинством данного CASE-инструмента в контексте решаемой задачи является возможность изменения структуры его репозитория, причем для этого не требуется писать какой-либо плагин – механизм предоставляется в готовом виде и работает в диалоговом режиме. Результатом его работы являются пользовательские расширения (user extensions) – дополнительные свойства или типы, которые добавляются в существующий репозиторий [Там же].

Репозиторий OD, как и любой другой репозиторий, имеет свое метаописание (или метасхему) – совокупность описаний типов, их свойств, а также связей между объектами этих типов [8]. Все структуры данных, находящихся в репозитории, можно разделить на три основных типа (метатипа), составляющих метамодель OD.

1. Элементные типы (англ. element types) – типы возможных объектов, которые могут содержаться в репозитории. Стандартными элементными типами OD являются функции, модули, сущности, таблицы, колонки и пр. Каждый элементный тип состоит из набора свойств – дискретных характеристик данного типа.

2. Ассоциативные типы (англ. association types) – типы возможных бинарных отношений между объектами тех или иных элементных типов. Стандартными ассоциативными типами OD являются использование сущностей в функциях, использование модулей организационными единицами и пр. Так же как и элементные типы, они состоят из набора определенных свойств.

3. Текстовые типы (англ. text types) – структуры данных для больших текстовых свойств элементного или ассоциативного типа. Стандартными текстовыми типами OD являются такие типы как описание, примечание и др. В диалоге текстовый тип представляет собой одну область, содержащую несколько строк с текстом. Текстовые типы отображаются в RON как свойства элементов и ассоциаций, но концептуально это отдельные типы [7].

Еще одним плюсом CASE-системы Oracle Designer является то, что при создании пользовательских расширений автоматически генерируется механизм доступа к их экземплярам. В данной CASE-системе эти механизмы носят название API. OD API – это набор представлений базы данных и пакетов языка программирования PL/SQL в схеме владельца репозитория, которые обеспечивают возможность безопасного доступа к данным репозитория. Тот факт, что OD API доступен для внесения изменений, в

последующем позволит внедрить на уровне этого интерфейса механизм проверки ERM-схем на непротиворечивость. Стоит отметить, что пакеты API – это единственный способ записи информации в репозиторий, и все инструментальные средства используют их независимо от того, входят эти средства в состав OD или нет [7].

4. Реализация ERM-репозитория как расширения OD-репозитория

Прежде чем определять в метасхеме OD-репозитория ERM-метасхему, решено было сначала реализовать ее в виде реляционной БД и проверить работоспособность полученных структур на реальных примерах ERM-схем и типичных транзакциях их изменения. Проектирование и реализация этой тестовой БД осуществлялись в соответствии с методикой разработки, реализованной в OD, – создание ER-схемы в нотации Баркера (рис. 1), автоматическая трансформация ее в реляционную схему, генерация и выполнение команд языка определения данных SQL. При этом в ходе реализации учитывались особенности механизма пользовательских расширений, в частности поддержка ассоциативных типов, имеющих атрибуты (аналоги в ER-модели Баркера – множества связей, атрибутов не имеют). В процессе построения ER-диаграммы такие структуры преобразовывались во множества сущностей.

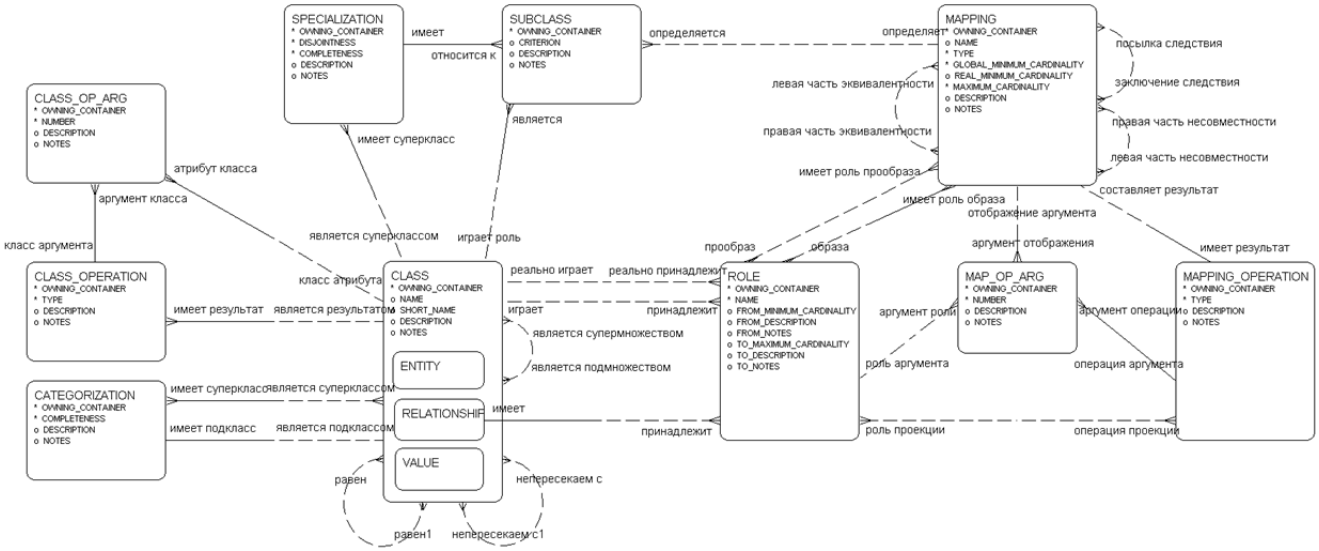

Рис. 1. ERM-метасхема в нотации Баркера

Полученные в ходе тестовой реализации отношения были преобразованы в наборы элементных и ассоциативных типов и успешно добавлены в метасхему репозитория с помощью утилиты Maintain User Extensions инструмента Repository Administration Utility [7]. Вновь созданные типы репозитория сразу же стали доступны в диалоге RON (рис. 2), что позволило заносить в репозиторий отладочные ERM-схемы. Автоматически сгенерированный API позволил сделать это в пакетном режиме массовой загрузки.

Как было отмечено ранее, при определении пользовательских расширений автоматически создаются API доступа к объектам этих типов. Одно из требований, предъявляемых к разрабатываемой системе, заключается в том, что должна быть обеспечена возможность работы с репозиторием ряда других, внешних приложений. Теоретически эти приложения могут работать и посредством стандартного генерируемого OD API, однако есть ряд недостатков, возникающих при этом подходе.

Первый недостаток заключается в том, что стандартный OD API имеет достаточно сложную структуру. Перед совершением непосредственно операции над объектом необходимо выполнить ряд вспомогательных действий, таких как указание текущего контекста, инициализация сессии, проверка транзакции и обработка ошибок. Все эти действия должны будут производиться внешним приложением. К тому же представления, через которые читаются данные об объектах, имеют достаточно сложные и запутанные связи с описаниями прикладных систем, в рамках которых они созданы. Все эти обстоятельства существенно усложнят процесс разработки нового или интеграции уже существующего стороннего приложения. Длительность разработки такого приложения также будет увеличена.

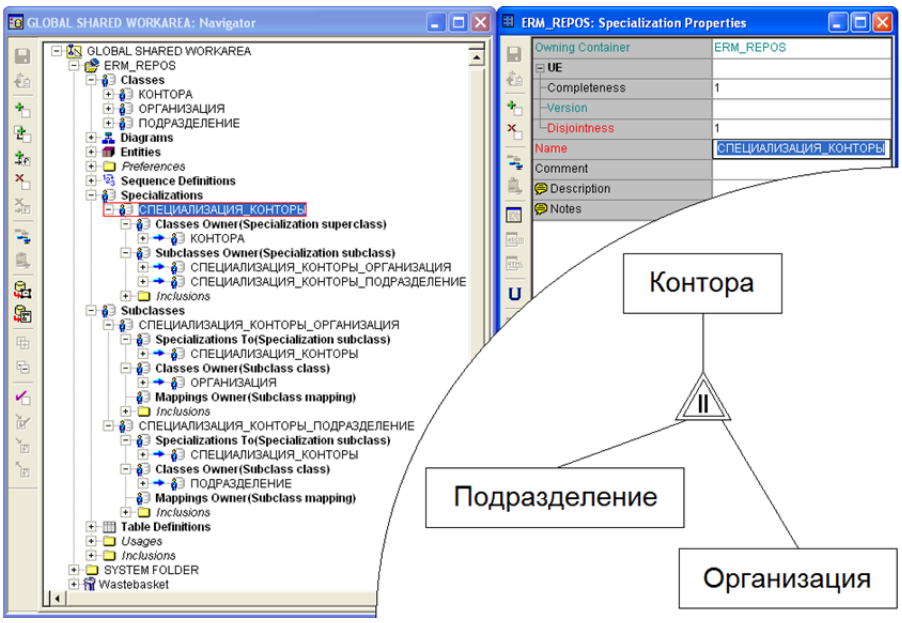

Рис. 2. Диалог RON по созданию показанного фрагмента ERM-схемы

Второй недостаток заключается в том, что логика работы с некоторыми сложными структурными сущностями ERM-модели выходит далеко за рамки простого добавления, изменения или удаления какого-либо объекта репозитория. Например, ассоциация множества сущностей с множеством связей с точки зрения графической нотации ERM-модели сопровождается соединением одним ребром соответствующих вершин графа. Однако в репозитории для каждого множества сущностей будут дополнительно созданы объект элементного типа «роль», который будет хранить соответствующие минимальные и максимальные кардинальные числа, и два объекта ассоциативных типов. Если внешнее приложение будет использовать генерируемое API, ответственность за создание и своевременное удаление всех этих объектов ляжет именно на него. Таким образом, логика работы со структурными объектами ERM-модели будет существенно усложнена и к тому же разбита по нескольким архитектурным слоям.

Третий недостаток заключается в том, что в условиях, когда логика зависимостей между структурными элементами ERM-модели распределена по разным слоям системы, существенно усложняется возможность внесения в эту логику каких-либо корректив. При изменении семантики некоторого структурного элемента ERM-модели (а такое не исключается в условиях эволюции модели) соответствующим изменениям подвергнется его обработка как на уровне репозитория, так и на уровне всех внешних приложений, его использующих, что усложнит процесс поддержки системы.

Учитывая вышесказанное, было принято решение обособить логику работы со структурными сущностями ERM-модели в отдельный пакет процедур и функций (ERM_API), осуществляющих добавление, изменение, удаление и выборку объектов этой модели. Механизм доступа инкапсулирует в себе обращение к репозиторию через OD API. Каждый вызов той или иной процедуры или функции механизма порождает один или более вызовов сгенерированного OD API, предварительную подготовку к ним, а также последующую обработку и возврат результатов (рис. 3).

Как уже было сказано, механизм доступа представляет собой пакет процедур и функций. Для каждого типа структурных сущностей ERM-модели есть четыре обязательные операции – добавление, изменение, удаление и выборка данных. Для некоторых типов структурных объектов созданы операции, предоставляющие доступ к дополнительной информации, связанной с объектом (например, для операции над классами – список классов-операндов и др.). Также в пакете присутствует ряд процедур, локализующих обращение к сгенерированному OD API.

Одной из ключевых в рамках работ по созданию репозитория является задача реализации механизма проверки ERM-схем на непротиворечивость. Необходимо, чтобы в любой момент времени аналитические и проектные данные, хранимые в репозитории, были целостными и непротиворечивыми.

Важным вопросом является расположение механизма проверки в системе. Для обеспечения целостности хранимой в репозитории информации проверка должна производиться при работе с любыми инструментами, осуществляющими изменения хранимых в репозитории данных. Работа внешних ERM-приложений осуществляется путем обращения к функциям созданного механизма доступа ERM_API, а вот инструмент RON обращается непосредственно к OD API. Напомним, что именно этот компонент OD применяется для диалоговой работы с ERM-репозиторием. Следовательно, расположение механизма проверки схем на непротиворечивость должно быть не выше уровня OD API.

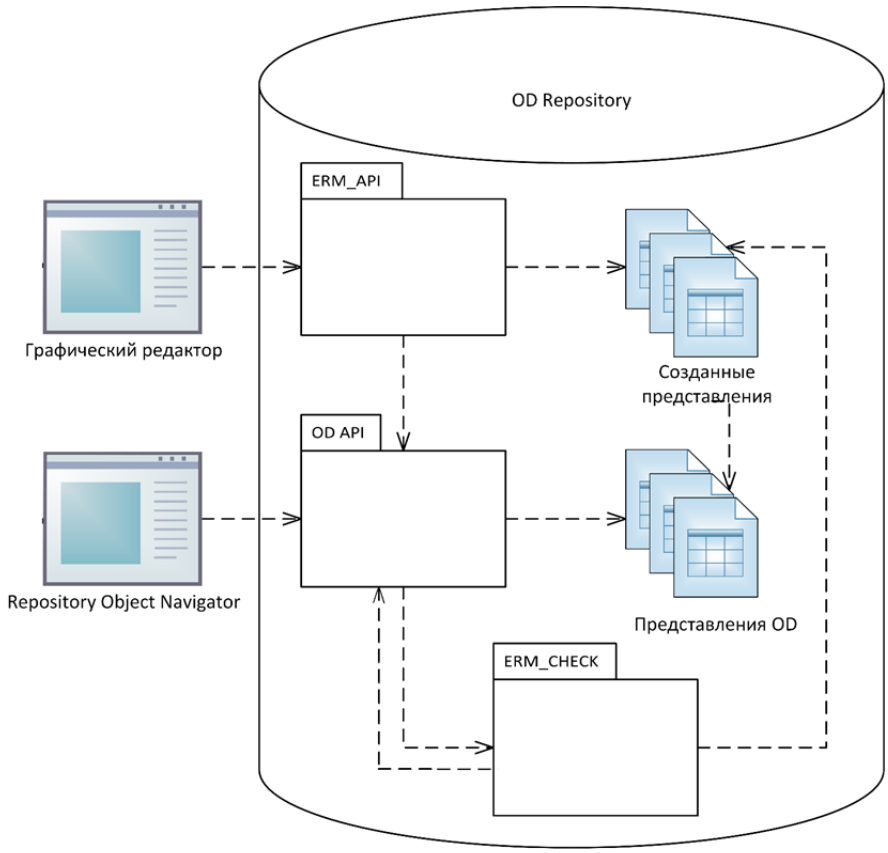

Рис. 3. Архитектура ERM-репозитория

Вышеуказанные обстоятельства заставили реализовать пакет функций проверок ERM-схем на непротиворечивость (ERM_CHECK) как расширение пакета OD API. Доступность исходного кода последнего позволила добавить в него обращения к функциям верификации схем репозитория. Таким образом, механизм проведет проверку изменений, совершенных как во внешнем инструменте, использующем OD API, так и в стандартных инструментах OD, таких как RON.

Подробнее о решении проблемы непротиворечивости схем данных в ERM-репозитории речь пойдет речь в последующих статьях авторов.

Заключение

В статье описаны принципы, архитектура и реализация прототипа ERM-репозитория. Его создание позволило развернуть тестирование и отладку реальной работы многих других подсистем исследовательского CASE-инструмента для ERM-модели, который должен стать практическим доказательством актуальности и работоспособности последней. Важнейшими из них являются графический редактор ERM-схем, реализующий методику ERM-моделирования, и максимально эффективный транслятор ERM-схем в реляционную модель, генерирующий помимо команд CREATE TABLE команды создания представлений и триггеров.

ЛИТЕРАТУРА

1. Halpin T., Morgan T. Information Modeling and Relational Databases, Second Edition. Morgan Kaufman, 2008.
2. Бабанов А.М. Семантическая модель «Сущность – Связь – Отображение» // Вестник Томского государственного университета. Управление, вычислительная техника и информатика. 2007. № 1. С. 77–91.
3. Agarwal B., Tayal S., Gupta M. Software Engineering and Testing. Jones & Bartlett Learning, 2010.
4. Hamid B. A model-driven approach for developing a model repository: Methodology and tool support // Future Generation Computer Systems. 2017. № 68. P. 453–490.
5. Бабанов А.М. Перспективы проектирования БД, открывающиеся с применением современных семантических моделей данных // Вестник Томского государственного университета. Управление, вычислительная техника и информатика. 2015. № 2. С. 73–80.
6. Welke R. The case repository: more than another database application // Challenges and strategies for research in systems development. New York : John Wiley & Sons, Inc., 1992. P. 181–218.
7. Колетски П., Дорси П. Oracle Designer. Настольная книга пользователя. М. : Лори, 1999.
8. Welke R. The case repository: more than another database application. Meta Systems, Ltd., 1988.

Бабанов Алексей Михайлович, канд. техн. наук, доцент. E-mail: babanov2000@mail.ru
Петров Александр Викторович. E-mail: flamingcode@mail.ru
Национальный исследовательский Томский государственный университет

Поступила в редакцию 20 апреля 2017 г.

Babanov Alexey M., Petrov Alexander V. (National Research Tomsk State University, Russian Federation).
Implementation of the ERM-model repository in CASE-system Oracle Designer.
Keywords: ERM-model; repository; CASE-system; Oracle Designer.

DOI: 10.17223/19988605/41/6

Recently researchers of the automated means for information system development pursue the goal to increase of automation degree, developer labor productivity, and also quality of the result received by means of CASE-systems. For achievement of these purposes it is necessary to improve analysis and design models, tools of intermodel transformation and final generation of executed artifacts. Testing and debugging of the newly proposed means are inconceivable without creation of a research prototype of the future CASE-system. This article is devoted to development of a kernel of such system - a repository. Distinctive feature of this development is orientation to use the semantic data model «Entity - Relationship - Mapping» (ERM-model) for information modeling.

Along with the traditional functional requirements demanded for automated development tools, the research prototype of a repository has a number of important nonfunctional requirements:

- the system should have the program interface allowing to work with schemes by means of external applications, such as the graphic editor of ERM-schemes, generator of relational schemes and other tools which will be realized in the future;
- the system should have the user interface giving dialogue access to objects of a repository;
- as the ERM-model actively develops, its metascheme will be frequently changed, hence, the architecture of a prototype should be flexible and give possibility to reuse of earlier developed components.

It is possible to distinguish two alternative ways of the ERM-repository realization. The first way consists in creation of own repository «from scratch» based on of permanent data store. Relational DBMS can acts as data store, for example. The second way consists in expansion of any already available CASE-mean. During the analysis of base means CASE-system Oracle Designer (OD) is taken as a platform for creation of the ERM-scheme repository.

The central part of this system is the repository containing project specifications at all its stages and providing coordinated work of all its participants, whatever roles they played in the project. For access to the repository and management of it Repository Object Navigator is available as well as specialized dialogue tools. It is a universal instrument, allowing looking through and modifying almost all objects stored in a repository.

Before defining the ERM-metascheme in the metascheme of an OD-repository, it has been decided to realize this metascheme in the form of a relational database at first, to check up working capacity of the received structures on real examples of the ERM-schemes and typical transactions of their change. The relations received during test realization have been transformed to sets of element and association types and successfully added in the metascheme of a repository by means of Maintain User Extensions utility of Repository Administration Utility tool.

During repository designing the decision to isolate logic of work with structural ERM-model entities in a separate package (ERM_API) was accepted. Its procedures and functions carry out adding, modifying, deleting and retrieving objects of this model. The access mechanism encapsulates calls to a repository through standard OD API. Each call of any procedure or function generates one or more calls of OD API, preliminary preparation for them, and the subsequent processing and return of results.

The problem of realization of the ERM-scheme checks on consistency is one of key on repository creation. It is important, that at any moment the analytical and design data, stored in the repository, were complete and consistent. For maintenance of check inevitability at any changes it has been decided to realize a package of the ERM-scheme check functions (ERM_CHECK) as expansion of package OD API, in appropriate way having changed the last.

Timely creation of the ERM-repository has allowed developing, testing and debugging of many other subsystems of the research CASE tool for ERM-model which should become the practical proof of an urgency and working capacity of the last. Major of these subsystems are the graphic editor of ERM-schemes realizing a technique of ERM-modeling, and as much as possible effective compiler of ERM-schemes in the relational model, generating besides CREATE TABLE commands of view and trigger creation.

REFERENCES

1. Halpin, T. & Morgan, T. (2008) *Information Modeling and Relational Databases*. 2nd ed. Morgan Kaufman.
2. Babanov, A. M. (2007) Semanticheskaya model' "Sushchnost' – Svyaz' – Otobrazhenie" [Semantic model 'Entity – Relationship – Mapping']. *Vestnik Tomskogo gosudarstvennogo universiteta. Upravlenie, vychislitel'naya tekhnika i informatika – Tomsk State University Journal of Control and Computer Science*. 1(1). pp. 77–91.
3. Agarwal, B., Tayal, S. & Gupta, M. (2010) *Software Engineering and Testing*. Jones & Bartlett Learning.
4. Hamid, B. (2017) A model-driven approach for developing a model repository: Methodology and tool support. *Future Generation Computer Systems*. 68. pp. 453–490. DOI: 10.1016/j.future.2016.04.018
5. Babanov, A. M. (2015) Database design prospects opening with application of modern semantic data models. *Vestnik Tomskogo gosudarstvennogo universiteta. Upravlenie, vychislitel'naya tekhnika i informatika – Tomsk State University Journal of Control and Computer Science.* 2. pp. 73–80. (In Russian). DOI: 10.17223/19988605/31/8
6. Welke, R. (1992) The case repository: more than another database application. In: Cotterman, W.W. & Senn, J.A. (eds) *Challenges and strategies for research in systems development*. New York: John Wiley & Sons. pp. 181–218.
7. Koletzke, P. & Dorsey, P. (1999) *Oracle Designer. Nastol'naya kniga pol'zovatelya* [Oracle Designer. Handbook]. Moscow: Lory.
8. Welke, R. (1988) *The case repository: more than another database application*. Meta Systems, Ltd.