Подход к разработке программных компонентов для формирования баз знаний на основе концептуальных моделей

И. В. Бычков, Н. О. Дородных, А. Ю. Юрин*

Институт динамики систем и теории управления им. В.М. Матросова СО РАН, Иркутск, Россия

*Контактный e-mail: iskander@icc.ru

Рассмотрен подход к автоматизации процесса разработки программных компонентов интеллектуальных систем, в частности компонентов, обеспечивающих автоматизированное формирование баз знаний на основе трансформации концептуальных моделей. Приведено описание основных элементов подхода: модели типового программного компонента; языка TMRL (Transformation Model Representation Language), предназначенного для представления и хранения модели трансляции; метода создания программных компонентов на основе "клонирования" типового программного компонента и его настройки (специализации); SaaS-архитектуры (Software as a Service, программное обеспечение как услуга) сервис-ориентированного программного средства.

Ключевые слова: автоматизация разработки программных компонентов, интеллектуальная система, концептуальная модель, база знаний, трансформация моделей.

Введение

В настоящее время разработка новых методов и подходов к созданию интеллектуальных систем (ИС) и их программных компонентов остается перспективной областью научных исследований. Основным элементом ИС является база знаний (БЗ), обеспечивающая как ввод, хранение, обработку знаний (в том числе закономерностей и взаимосвязей какой-либо предметной области) по запросам пользователей, так и проведение так называемых сервисных функций — проверку на полноту, непротиворечивость и т. п.

Процесс формирования БЗ традиционно считают "узким местом" при разработке ИС и связывают с решением задач моделирования предметной области, идентификации (получения), концептуализации (структурирования) и формализации (представления) знаний с последующей программной реализацией, т. е. представлением на определенном языке представления знаний (ЯПЗ) [1, 2]. Сложность этих задач возрастает при необходимости обеспечения удаленного и распределенного доступа, согласования мнений экспертов (специалистов), а также использования информации, представленной в разных формах, в том числе в форме концептуальных моделей.

Концептуальные модели предметных областей создаются как в процессе разработки программного обеспечения, так и в процессе целенаправленного моделирования предметной области (построения ее онтологии или бизнес-модели). При создании моделей используют различные нотации, языки и программные средства (системы концептуального, когнитивного, онтологического моделирования, CASE-средства). Большинство из них, обеспечивая создание концептуальных моделей и генерацию документации, предусматривают ограниченные возможности по преобразованию построенных моделей в структуры ЯПЗ (частичное/скелетное преобразование), что, в свою очередь, затрудняет практическое использование построенных моделей для формирования баз знаний при разработке ИС.

В связи с этим актуально создание расширяемой сервис-ориентированной программной системы, обеспечивающей проектирование и синтез кода БЗ на определенном ЯПЗ путем трансформации концептуальных моделей. При этом под расширяемостью понимается возможность создания пользователем новых программных компонентов, реализующих функции анализа концептуальных моделей и генерации кода БЗ для других форматов и ЯПЗ.

В настоящей статье рассмотрены вопросы как разработки подхода к автоматизации создания программных компонентов, предназначенных для формирования БЗ на основе концептуальных моделей, так и реализации данного подхода в форме расширяемой веб-ориентированной системы.

1. Состояние вопроса

Проблема повышения эффективности разработки ИС и их программных компонентов является актуальной и решается в различных направлениях: от совершенствования методов разработки до создания специального программного инструментария.

Примером программного инструментария может служить программный комплекс АТ-ТЕХНОЛОГИЯ [3], обеспечивающий создание интегрированных экспертных систем (ЭС). Создаются новые технологии поддержки процесса разработки интеллектуальных СППР в слабоформализованных предметных областях [4]. Ведутся работы по созданию и использованию различных интеллектуальных веб-сервисов. Примером может служить Интернет-комплекс (платформа) IACPaaS, поддерживающий единые технологические принципы разработки прикладных и инструментальных ИС и управления ими [5]. Особое место занимают использование информационных моделей и их трансформация для создания интеллектуальных систем, в частности подхода, известного как модельно-управляемая разработка (MDD, Model Driven Development) [6 – 9]. В рамках исследовательского проекта HeKatE (Hybrid Knowledge Engineering) создана методология проектирования ИС для управления и поддержки принятия решений, включающая визуальную разработку БЗ и генерацию прототипа ИС, при этом используется визуальный формализм представления правил — XTT2 (eXtended Tabular Trees) [10], и др.

Как правило, исследователи предлагают комплексное решение проблемы автоматизации создания ИС и БЗ на основе использования различных подходов и методик: интегрированной модели представления знаний, онтологического подхода [4], специальных нотаций (языков) [6, 10], технологий облачных вычислений [5] и др. Также применяются различные способы приобретения знаний (извлечение знаний из экспертов, текстов, баз данных и др.), но при этом актуальными остаются вопросы обеспечения

распределенного характера процесса разработки БЗ и использования различных видов концептуальных моделей для его автоматизации.

Однако в целом проблема создания технологий (методологий/подхода) для поддержки разработки программных компонентов ИС, обеспечивающих автоматизированную разработку БЗ на основе трансформации концептуальных моделей предметных областей, требует проведения дополнительных исследований.

Необходимо отметить, что задача повторного использования концептуальных моделей может быть сведена к задаче трансформации моделей. В общем случае трансформация моделей представляет собой процесс автоматической генерации целевой модели по исходной модели в соответствии с набором правил трансформации (преобразования). Под правилом преобразования подразумевается описание того, как одна или более конструкций на исходном языке моделирования может быть преобразована в одну или более конструкций на целевом языке моделирования [11].

Трансформация моделей является одной из основных составляющих модельно-ориентированного подхода к разработке программного обеспечения Model Driven Engineering (MDE) [11]. Можно выделить несколько основных направлений к реализации трансформации моделей:

- преобразование с использованием графовых грамматик (Graph Rewriting) [12];
- трансформация на основе установления соответствий между конструкциями исходной и целевой метамодели;
- преобразование с использованием языков (стандартов) перевода XML-документов.

В рамках каждого направления создаются различные языки трансформации моделей. Большинство из этих языков развиваются в академическом сообществе. Наиболее распространенными языками трансформации моделей являются: QVT (Query/View/Transformation) [13], ATL (ATLAS Transformation Language) [14], VIATRA2 (VIsual Automated model TRAnsformations) [15], GReAT (Graph REwriting And Transformation) [16], XSLT (eXtensible Stylesheet Language Transformations) [17]. Их основной недостаток — высокие требования к специалисту (эксперту) при разработке правил (сценария) трансформаций. В частности, специалисту необходимо знать не только синтаксис и семантику определенного языка трансформации моделей (который достаточно сложен), но и языки метамоделирования, используемые для описания входной и выходной моделей. Следует также отметить, что программные средства, поддерживающие данные языки, зачастую не предоставляют возможность визуализации процесса описания правил трансформации моделей, т. е. правила преобразований создаются в специальных текстовых редакторах, ориентированных на программистов. Совокупность этих факторов затрудняет практическое использование этих языков и программных средств непрограммирующими специалистами-предметниками.

Тем не менее существует ряд работ, направленных на решение задачи повышения эффективности разработки БЗ путем повторного использования (трансформации) различных концептуальных моделей. Так, в работе [18] предлагается метамодельный управляемый подход к трансформации моделей для обмена правилами между онтологиями OWL/SWRL и диаграммами UML/OCL. В качестве промежуточного представления знаний (правил) используется язык REWERSE Rule Markup Language (R2ML), а для описания правил трансформации — язык трансформации моделей ATL. В [19] исследователи представляют подход трансформации диаграмм классов UML в онтологии OWL с использованием языка QVT. Другими успешными примерами решения задачи автоматизированной разработки БЗ путем трансформации моделей являются: преоб-

разование диаграмм UML в логические выражения [20, 21], преобразование диаграмм UML в онтологии OWL [22 – 24], преобразование диаграмм классов UML в конструкции ЯПЗ CLIPS [25, 26], преобразование онтологий и правил OWL/SWRL в конструкции ЯПЗ JESS [27 – 29], преобразование концепт-карт в онтологии OWL [30].

Особенностью рассмотренных исследований является узкая специализация в части поддержки форматов концептуальных моделей, ЯПЗ, языков трансформации моделей, что в свою очередь обусловливает разнообразие программного обеспечения, используемого исследователями в рамках каждого отдельного преобразования (иногда на каждом этапе преобразования используется отдельное специализированное средство). Соответственно, затруднительно сделать вывод о наличии единых принципов построения систем для разработки БЗ на основе концептуальных моделей. Это подтверждает актуальность разработки технологии и программного средства автоматизации создания программных компонентов для поддержки разработки БЗ на основе трансформации концептуальных моделей.

Одним из перспективных подходов к созданию программных средств с распределенным доступом является концепция “облачных вычислений” (Cloud Computing) [31]. Предлагается применить ее элементы, в частности SaaS-архитектуру (Software as a Service, программное обеспечение как услуга), при создании программного средства, реализующего разрабатываемый подход.

В качестве источников концептуальных моделей предлагается использовать группу моделей, для описания структуры которых используется XML (eXtensible Markup Language). Примерами моделей данной группы являются: диаграммы классов UML (Unified Modeling Language) [32], представленные в соответствии со стандартом XMI (XML Metadata Interchange) [33] — стандартом обмена информацией о моделях; концепт-карты, представленные в соответствии со стандартом XTM (XML Topic Maps) [34] и др. В качестве целевого ЯПЗ выбраны CLIPS [35] и OWL [36].

2. Элементы подхода

Разработана технология автоматизации процесса создания программных компонентов, обеспечивающих формирование БЗ путем трансформации концептуальных моделей предметных областей.

Основными элементами технологии являются:

- модель типового программного компонента, включающая модель трансляции;
- предметно-ориентированный (декларативный) язык TMRL (Transformation Model Representation Language), предназначенный для представления и хранения модели трансляции;
- метод создания программных компонентов на основе “клонирования” типового программного компонента и его настройки (специализации);
- концептуальная архитектура сервис-ориентированной системы и ее основных элементов, основанных на принципах SaaS.

Для программной реализации данной технологии автоматизации разрабатывается специализированная сервис-ориентированная система. Это веб-ориентированное прикладное программное обеспечение, которое представляет собой совокупность специализированных программных компонентов (веб-приложений), обеспечивающую решение задачи разработки БЗ информационных систем, а также возможность ее расширения

пользователем с помощью предлагаемого подхода. Программный компонент — это прикладной программный модуль (веб-сервис), предназначенный для автоматизированного создания БЗ (прототипов БЗ) на целевом ЯПЗ путем трансформации исходных информационных моделей, построенных при помощи различных систем концептуального, когнитивного, онтологического моделирования или CASE-средств, обеспечивающих представление моделей в виде XML. Основным преимуществом сервис-ориентированной системы является ее расширяемость. Создаваемые программные компоненты (сервисы) могут отчуждаться и использоваться в качестве автономных подсистем для других ИС.

Отдельные элементы указанного подхода рассмотрим подробнее.

3. Модель типового программного компонента

Для повышения эффективности разработки программных компонентов предложено использовать оригинальную модель типового программного компонента, представленную следующим образом:

$$M_{TPC} = \langle M_T, A_{IN}, CG_{OUT} \rangle , \tag{1}$$

где M_T — модель трансляции; A_{IN} — анализатор входных моделей (исходных концептуальных моделей); CG_{OUT} — генератор выходных моделей (кода БЗ на целевом ЯПЗ).

В процессе создания программного компонента (специализации модели типового программного компонента M_{TPC}) пользователю необходимо сформировать модель трансляции M_T, которая определяет правила преобразования исходных концептуальных моделей в целевые БЗ.

3.1. Модель трансляции

Используя (1), определим модель трансляции M_T:

$$M_T = \langle MM_{IN}, MM_{OUT}, T \rangle , \tag{2}$$

где MM_{IN} — метамодель исходной (входной) концептуальной модели; MM_{OUT} — метамодель целевой (выходной) модели представления знаний; T — оператор преобразования моделей.

Используя (2), подробнее опишем элементы M_T:

$$MM_{IN} = \langle E_{IN}, R_{IN} \rangle , \tag{3}$$

где E_{IN} — множество элементов метамодели исходной концептуальной модели; R_{IN} — множество отношений между элементами этой метамодели.

Так,

$$E_{IN} = \left\{ e_1^{in} ... e_n^{in} \right\}, \quad e_i^{in} = \langle id_i, name_i \rangle , \quad i \in \overline{1,n},$$

где id_i — идентификатор i-го элемента; $name_i$ — наименование i-го элемента или ссылка на другой элемент e_{link}^{in}, при котором $e_{link}^{in} = \langle id_{link}, name_{link} \rangle$, где id_{link} — идентификатор элемента; $name_{link}$ — наименование элемента.

$$R_{IN} = \left\{ r_1^{in} ... r_m^{in} \right\}, \quad r_j^{in} = \left\langle r_{LHS_j}^{in}, \quad r_{RHS_j}^{in} \right\rangle , \quad j \in \overline{1,m},$$

где $r_{LHS_j}^{in}$ — левая часть связи; $r_{RHS_j}^{in}$ — правая часть связи. В свою очередь, $r_{LHS_j}^{in} = e_i^{in}$ и $r_{RHS_j}^{in} = e_i^{in}$, где e_i^{in} — ссылка на элемент.

$$MM_{OUT} = \langle E_{OUT}, R_{OUT} \rangle , \tag{4}$$

где E_{OUT} — множество элементов метамодели целевой БЗ; R_{OUT} — множество отношений между элементами метамодели целевой БЗ.

$$E_{OUT} = \left\{ e_1^{out} ... e_n^{out} \right\}, \quad e_i^{out} = \langle id_i, name_i \rangle , \quad i \in \overline{1,n},$$

где id_i — идентификатор i-го элемента; $name_i$ — наименование i-го элемента.

$$R_{OUT} = \left\{ r_1^{out} ... r_m^{out} \right\}, \quad r_j^{out} = \left\langle r_{LHS_j}^{out}, r_{RHS_j}^{out} \right\rangle , \quad j \in \overline{1,m},$$

где $r_{LHS_j}^{out}$ — левая часть связи; $r_{RHS_j}^{out}$ — правая часть связи. В свою очередь, $r_{LHS_j}^{out} = e_i^{out}$ и $r_{RHS_j}^{out} = e_i^{out}$, где e_i^{out} — ссылка на элемент.

Основным элементом модели трансляции M_T является оператор преобразования T:

$$T : CM \to KB, \tag{5}$$

где CM — исходная концептуальная модель; KB — целевая БЗ.

Для уточнения (5) выделим виды синтаксиса концептуальных языков моделирования и ЯПЗ согласно [37]:

- Абстрактный синтаксис (abstract syntax) — определяет важнейшие концепции и структуру выражений языка, т. е. характеризует в абстрактной форме виды элементов, из которых состоит язык, и правила их комбинирования. Использование метамоделей (метамоделирование) — один из основных подходов к определению абстрактного синтаксиса языков, в том числе концептуальных языков моделирования и ЯПЗ.
- Конкретный синтаксис (concrete syntax) — определяет, как языковые элементы проявляются в конкретной, используемой человеком форме (текстовой или графической), т. е. конкретный синтаксис является нотацией. Абстрактный синтаксис напрямую связан с семантикой, а конкретный синтаксис — с абстрактным синтаксисом.
- Синтаксис сериализации или служебный синтаксис (serialization syntax) — подобен конкретному синтаксису, за исключением того, что он не обязан быть воспринимаемым человеком, т. е. данный синтаксис используется для хранения и обмена языковыми выражениями в сериализованной (последовательной) форме между различными программными средствами.

Специализируем (5) на абстрактном и конкретном (служебном) уровне:

$$T_{AS} : MM_{CM} \to MM_{KB}, T_{CS} : M_{XML} \to Code_{KRL}. \tag{6}$$

Здесь MM_{CM} — метамодель исходной концептуальной модели; MM_{KB} — метамодель целевой БЗ; M_{XML} — исходная концептуальная модель, представленная в формате XML; $Code_{KRL}$ — код БЗ, представленной на целевом ЯПЗ.

При этом $Code_{KRL} \in \{Code_{CLIPS}, Code_{OWL}\}$, где $Code_{CLIPS}$ — целевой код БЗ на ЯПЗ CLIPS; $Code_{OWL}$ — целевой код БЗ на ЯПЗ OWL.

Согласно (6), процесс трансформации M_{XML} в $Code_{KRL}$ осуществляется путем установления соответствий (определения правил трансформации) между абстрактными элементами исходной MM_{CM} и целевой MM_{KB} метамоделями.

Таким образом, для обозначения набора правил соответствия элементов метамоделей (правил трансформации) введем оператор

$$R_T = (r_1, r_2, ..., r_n) : MM_{CM} \to MM_{KB}, \tag{7}$$

где r_i — правило трансформации, представляемое в форме продукции. Здесь $r_i = \langle e_i^{in}, e_i^{out}, p_i \rangle, i \in \overline{1,n}$, где e_i^{in} — исходный элемент метамодели MM_{CM} (левая часть правила); e_i^{out} — целевой элемент метамодели MM_{KB} (правая часть правила); p_i — приоритет выполнения правила, определяющий последовательность выполнения правил, $p_i \in \overline{1,k}$.

4. Предметно-ориентированный язык TMRL

Для представления и хранения модели трансляции M_T разработан предметно-ориентированный язык (DSL) — Transformation Model Representation Language (TMRL). Грамматика разработанного TMRL принадлежит к классу контекстно-свободных грамматик (КС-грамматик, LL(1)) [38]. Конструкции TMRL позволяют в декларативном виде описывать элементы модели трансляции, в частности правила соответствия элементов метамоделей R_T.

Опишем основные конструкции TMRL с использованием расширенной формы Бэкуса — Наура (нотации РБНФ). Для упрощения восприятия описания TMRL опустим некоторые нетерминальные символы.

Модель трансляции M_T, описанная на TMRL, имеет следующий вид:

```
Модель трансляции на TMRL = Исходная метамодель, Целевая метамодель,
   Оператор трансформации;
Исходная метамодель = "Source Meta-Model", Заголовок, Тело метамодели;
Целевая метамодель = "Target Meta-Model", Заголовок, Тело метамодели;
Оператор трансформации = "Transformation", Заголовок, Тело трансформации;
Заголовок = Буква, { Буква | Цифра | Символ };
Тело метамодели = Элементы, Связи ;
Элементы = "Elements", "(" , { Элемент } , ")" ;
Элемент = Наименование, [ Атрибуты ] , "," ;
Атрибуты = "attributes", "(" , { Атрибут } , ")" ;
Атрибут = Наименование , "," ;
Наименование = Буква , { Буква | Цифра | Символ } ;
Связи = "Relationships" , "(" , { Связь } , ")" ;
Связь = Ассоциация | Связь по идентификатору ;
Ассоциация = Левый элемент ассоциации , "is associated with" ,
   Правый элемент ассоциации , "," ;
Левый элемент ассоциации = Наименование ;
Правый элемент ассоциации = Наименование ;
Связь по идентификатору = Левый элемент связи по идентификатору, "is" ,
   Правый элемент связи по идентификатору , "," ;
Левый элемент связи по идентификатору = Наименование элемента , "(" ,
   Наименование атрибута , ")" ;
Правый элемент связи по идентификатору = Наименование элемента , "(" ,
   Наименование атрибута , ")" ;
Наименование элемента = Наименование ;
```

```
Наименование атрибута = Наименование ;
Тело трансформации = { Правило } ;
Правило = Заголовок правила , Тело правила ;
Заголовок правила = "Rule" , Исходный элемент , "to" , Целевые элементы ,
   "priority" , Цифра ;
Исходный элемент = Наименование ;
Целевые элементы = Целевой элемент , { Дополнительный целевой элемент } ;
Дополнительный целевой элемент = "or" , Целевой элемент ;
Целевой элемент = Наименование ;
Тело правила = { Выражение определения } , { Условное выражение } ;
Выражение определения = Левый элемент выражения , "is" ,
   Правый элемент выражения ;
Левый элемент выражения = Исходный элемент ,[ Атрибут исходного элемента ];
Атрибут исходного элемента = Наименование ;
Атрибут исходного элемента = "(" , Наименование , ")" ;
Правый элемент выражения = Целевой элемент , [ Атрибут целевого элемента ] ,
   [ Дополнительный правый элемент выражения ] ;
Атрибут целевого элемента = "(" , Наименование , ")" ;
Дополнительный правый элемент выражения = "or" Целевой элемент ,
   [ Атрибут целевого элемента ] ;
Условное выражение = "if" , "(" , Условие , ")" , Действие ;
Условие = Одинарное условие , { Дополнительное условие } ;
Одинарное условие = Элемент условия , "is", Значения ;
Дополнительное условие = "and" , Одинарное условие ;
Элемент условия = Исходный элемент , [ Атрибут исходного элемента ] ;
Значения = Значение | Множество значений ;
Значение = "‘" , { Буква | Цифра | Символы } , "’" ;
Множество значений = "(" , Значение , { Дополнительное значение } , ")" ;
Дополнительное значение = "or" , Значение ;
Действие = Целевой элемент , Атрибут целевого элемента , "is" , Значение ;
```

Приведем пример фрагмента модели трансляции M_T, представленной на языке TMRL и описывающей преобразование диаграммы классов UML в элементы онтологической модели [26]. Модель трансляции состоит из трех блоков:

1. Описание исходной метамодели (элементы и отношения):

```
Source Meta-Model UML-XSD
    Elements (
        Model,
        Class attributes (xmi.id,    name),
        ...
    )
    Relationships (
        Model is associated with Namespace.ownedElement,
        Namespace.ownedElement is associated with Class,
        DataType(xmi.id) is Attribute(type),
        ...
    )
```

Блок исходной метамодели „Source Meta-Model“ с наименованием „UML-XSD“ содержит описание метамодели UML (диаграммы классов), представленной в формате XSD, включая элементы метамодели (раздел „Elements“). В данном примере это элементы „Model“ и „Class“. Заметим, элемент „Class‘ обладает свойствами „xmi.id“ и „name“. Помимо описания элементов исходная метамодель содержит описание связей между ее элементами, в том числе по идентификаторам, например связь атрибута с типом данных: „DataType(xmi.id) is Attribute(type)“. Описание блока на TMRL формируется автоматически в результате анализа исходной модели (этапы 2.2 и 2.4 алгоритма создания модели трансляции).

2. Описание целевой метамодели:

```
Target Meta-Model ONT
    Elements (
        ExtendedOntology attributes (id, name),
        Class attributes (id, name),
        ...
    )
    Relationships (
        Ontology is associated with Class,
        ...
    )
```

Блок целевой метамодели „Target Meta-Model“ с наименованием „ONT“ содержит описание модели онтологии. Структура блока аналогична структуре блока исходной метамодели, в частности, раздел „Elements“ содержит описание элементов целевой метамодели, а раздел „Relationships“ — связей между этими элементами. Описание блока на TMRL формируется автоматически (этап 2.4 алгоритма создания модели трансляции).

3. Описание правил преобразования:

```
Transformation UML-XSD to ONT
    Rule Model to Ontology priority 1
        Ontology(name) is Model or ModelElement.name
        Ontology(id) is Model(xmi.id)
    Rule Namespace.ownedElement-Class to Class priority 2
        Class(name) is Namespace.ownedElement-Class or
Namespace.ownedElement-Class(name) or ModelElement.name
        Class(id) is Namespace.ownedElement-Class(xmi.id)
    ...
    Rule AssociationEnd.multiplicity-Multiplicity-Multiplicity.
range-MultiplicityRange to Operator priority 7
        if ( MultiplicityRange(lower) is (‘‘1’’ or ‘‘-1’’) and
MultiplicityRange(upper) is (‘‘1’’ or ‘‘-1’’) )
            Operator(name) is ‘‘AND’’
    ...
```

Блок описания трансформаций „Transformation“ содержит описание правил преобразования (отображения) элементов исходной „UML-XSD“ и целевой „ONT“ метамоделей. Каждое правило помимо описания возможных соответствий элементов содержит

также указание на приоритет выполнения правила „priority“. В приведенном примере рассмотрены правила отображения между элементами „Model“ и „Ontology“, а также „Model-Class“ и „Ontology-Class“. Описание данных правил на TMRL формируется автоматически (этапы 2.3 и 2.4 алгоритма создания модели трансляции). В создаваемых правилах могут использоваться логические операторы „and“ или „or“, а также оператор условного выбора „if“, в частности, с целью задания определенного значения (константы) сущности метамодели целевой онтологической модели в соответствии с выполнением условия, определенного в данном блоке. В приведенном примере задается значение „AND“ элементу онтологической модели „Operator“, если кратность связи в UML-модели „MultiplicityRange“ равна диапазону „1-N“. Привила подобного вида задаются на этапе 2.5 алгоритма создания модели трансляции.

5. Метод создания программных компонентов

Метод создания программных компонентов представляет собой систематизированную совокупность действий, которые нацелены на решение задачи автоматизированной разработки программного компонента на основе “клонирования” (копирования) и настройки типового программного компонента (специализации модели типового программного компонента M_{TPC} путем формирования модели трансляции M_T) при помощи специального сервиса разработки программных компонентов, входящего в состав сервис-ориентированной системы.

Основные принципы предлагаемого метода создания программных компонентов:

- визуальное построение правил трансформации (соответствий) элементов метамодели исходной концептуальной модели в элементы метамодели целевой БЗ без использования специализированных языков трансформации моделей общего назначения (например, QVT, ATL и др.);
- автоматическое выделение элементов входных метамоделей для процесса визуального построения правил трансформации на основе анализа XSD-структур;
- представление и хранение установленных правил соответствия элементов метамоделей с использованием предметно-ориентированного (декларативного) языка TMRL.

Ограничения на входные и выходные метамодели:

- Метамодель исходных концептуальных моделей MM_{CM} должна быть представлена в формате XML-схемы (XML Schema) — XML Schema Definition (XSD) [39].
- Для автоматического построения XML-схем разработаны многочисленные паттерны (шаблоны) проектирования, наиболее распространенными из которых являются Russian Doll, Salami Slice, Venetian Blind, Garden of Eden. Данные паттерны различаются количеством используемых глобальных элементов и типов. Предполагается, что использование паттернов Venetian Blind и Garden of Eden [40] наиболее перспективно.
- Метамодель может быть семантически некорректной, если исходная концептуальная модель составлена неверно (например, если в исходной модели наименования тегов отражают семантику предметной области, а не метамодели).
- Не все связи элементов могут быть отражены на уровне XSD-метамодели, т. е. связь может быть установлена путем внутренней индексации элементов (уровень модели).

- Не все элементы XSD-метамодели или их значения могут быть однозначно интерпретированы в элементы (значения) целевой метамодели БЗ.
- Метамодели целевых БЗ MM_{KB}, представленных на ЯПЗ CLIPS и OWL, доступны для построения модели трансляции M_T по умолчанию.

Основные этапы метода создания программных компонентов на основе модели программного компонента ("клонирование" типового программного компонента) и настройки модели трансляции представлены на рис. 1 и 2.

Этапы создания программного компонента рассмотрим подробнее.

На этапе 1 пользователь средствами внешних программ формирует XSD-метамодель, которая соответствует исходной концептуальной модели. Затем данная метамодель загружается на сервис разработки программных компонентов, входящий в состав сервис-ориентированной системы.

На этапе 2 пользователь средствами сервис-ориентированной системы (сервис разработки программных компонентов) формирует модель трансляции M_T.

На этапе 2.1 процесса автоматизированного анализа XML-структуры метамодели XSD извлекаются элементы исходной метамодели MM_{CM}.

На этапе 2.2 на основе выделенных элементов метамодели MM_{CM} автоматически формируется предварительная модель трансляции M_T, содержащая только наборы элементов исходной и целевой метамоделей и не включающая правила трансформации.

На этапе 2.3 при помощи специального модуля визуального формирования соответствий элементов, входящего в состав сервиса разработки программных компонентов, устанавливаются правила трансформации исходных элементов метамодели MM_{CM} в целевые элементы метамодели MM_{KB}.

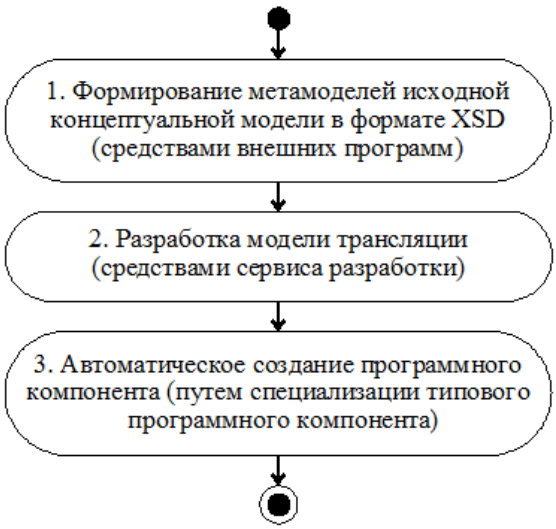

Рис. 1. Этапы создания программного компонента

Рис. 2. Этапы создания модели трансляции

На этапе 2.4 автоматически генерируется модель трансляции M_T на TMRL. Необходимо отметить, что данная модель может быть неполной (некорректной) по двум основным причинам.

1. Неполнота представления исходной модели: не все связи элементов концептуальной модели могут быть отражены на уровне XSD-метамодели MM_{CM}, что обусловлено особенностями механизма установления связи между элементами концептуальной модели:

- Связь элементов может осуществляться путем вложенности элементов друг в друга. Пример связи класса с атрибутом:
```
<Class ... >
    <Attribute ...> ... </Attribute>
</Class>
```
- Связь элементов может осуществляться путем указания ссылок на идентификаторы элементов. Пример связи атрибута класса с типом данных:
```
<DataType id=''номер типа данных''... />
...
<Attribute type=''номер типа данных'' ...> ... </UML:Attribute>
```

Связь элементов, установленная с использованием уникальных идентификаторов, не будет отображена на XML-схеме, вследствие чего при автоматической генерации модель трансляции M_T будет неполной (некорректной). Это является одним из недостатков использования XML-схем в качестве исходных (входных) метамоделей.

2. Неоднозначность интерпретации исходной модели: не все элементы XSD-метамодели MM_{CM} могут быть однозначно интерпретированы и отображены в элементы целевой метамодели БЗ MM_{KB}. Возможны ситуации, когда в качестве значения целевого элемента метамодели MM_{KB} используется текстовая последовательность (константа), значение которой, в свою очередь, может зависеть от значений исходных элементов метамодели MM_{CM}.

Исходя из этого, правила трансформации элементов можно разделить на три типа:

- правила определения — задают соответствия между исходными и целевыми элементами (их значениями) метамодели (определяются на этапе 2.3);
- правила связи — задают отношения между исходными элементами метамодели по ссылкам;
- составные правила определения — задают значения (константы) целевых элементов исходя из соответствующих значений исходных элементов.

Таким образом, установленные правила трансформации исходных и целевых элементов метамодели (этап 2.3) могут быть недостаточными для полного определения модели трансляции M_T.

Для решения данной проблемы на этапе 2.5 модель трансляции M_T уточняется (дополняется), т. е. определяются недостающие правила трансформации (правила связи и составные правила определения), которые не были установлены на этапе 2.3. Уточнение осуществляется при помощи специального текстового редактора для TMRL, входящего в состав сервиса. Результатом данного этапа является окончательная модель трансляции M_T, описывающая трансформацию исходных концептуальных моделей в код БЗ на целевых ЯПЗ.

На этапе 3 автоматически создается программный компонент путем специализации типового программного компонента на основе сформированной модели трансляции M_T.

6. Концептуальная архитектура системы

С целью программной реализации данного подхода исследовалась и специализировалась модель “Программное обеспечение как услуга” (Software-as-a-Service — SaaS) [31].

Приведем формальную постановку данной задачи: в общем виде специализированная SaaS-модель для сервис-ориентированной системы может быть описана как

$$SaaS = \langle R, S, A \rangle , \tag{8}$$

где R — набор предоставляемых сервис-ориентированной системой ресурсов (услуг) пользователям; S — набор сервисов (подсистем), предоставляющий возможность повсеместного и удобного сетевого доступа к ресурсам сервис-ориентированной системой R; A — аппаратное обеспечение как вычислительные узлы (основа) для размещения программного обеспечения S.

При этом $S = \langle S_{SYS}, S_{USR}, I \rangle$, где S_{SYS} — множество системных сервисов, обеспечивающих базовое взаимодействие пользователей с сервис-ориентированной системой и предоставляющих возможность разработки и регистрации (развертывания) пользовательских прикладных сервисов (программных компонентов); S_{USR} — множество разработанных и зарегистрированных программных компонентов в составе сервис-ориентированной системы; I — интерфейс взаимодействия с внешними ИС.

В свою очередь, $I = \{i_1 ... i_n\}, i_j = \langle name_j, command_j \rangle, j \in \overline{1, n}$, где $name_j$ — наименование j-го метода взаимодействия; $command_j$ — управляющая команда j-го метода взаимодействия.

Используя (8), уточним множество предоставляемых сервис-ориентированной системой ресурсов (функций) R для набора системных сервисов S_{SYS}:

- разработка программного компонента на основе модели программного компонента M_{TPC}, включая визуальное и текстовое построение соответствий (правил трансформации) между элементами исходной и целевой метамоделями (создание модели трансляции M_T);
- хранение полученных из концептуальных моделей знаний с использованием специальной модели расширенной онтологии и продукционной модели [26];
- визуальное отображение и редактирование понятий предметной области и их отношений в виде графа (онтологической модели);
- визуальное отображение и редактирование (моделирование) продукций с использованием специальной нотации — Rule Visual Modeling Language (RVML) [41].

Используя (8), уточним множество предоставляемых сервис-ориентированной системой ресурсов (функций) R для набора прикладных сервисов S_{USR}:

- формирование онтологической модели на основе автоматизированного анализа концептуальных моделей предметных областей;
- генерация кода БЗ на целевом ЯПЗ путем автоматической трансформации модели онтологии;
- генерация кода БЗ на целевом ЯПЗ на основе прямой автоматической трансформации концептуальных моделей предметных областей.

Таким образом, концептуальная архитектура сервис-ориентированной системы позволяет описать ее структуру, включая состав и типы элементов, а также принципиальные особенности функционирования (рис. 3).

Рис. 3. Концептуальная архитектура сервис-ориентированной системы

Основные типы элементов:

- информационные ресурсы — предназначены для хранения служебной информации, которая используется системными и прикладными компонентами (подсистемами) сервис-ориентированной системы как для обеспечения собственного функционирования, так и для решения задач автоматизированного формирования БЗ;
- системные сервисы — представляют собой набор всех предлагаемых пользователям сервисов (подсистем), обеспечивающих их базовое взаимодействие с сервис-ориентированной системой и предоставляющих инструментарий для создания прикладных программных компонентов на основе типового;
- прикладные сервисы — представляют собой набор разработанных пользователями программных компонентов, обеспечивающих возможность автоматического синтеза БЗ путем трансформации концептуальных моделей. Программные компоненты создаются на основе типового программного компонента путем его “клонирования” и настройки (специализации).

В зависимости от решаемых задач и типов входных данных (исходных концептуальных моделей или модели онтологии) на основе типового программного компонента могут быть созданы разные виды программных компонентов. Приведем их классификацию по различным основаниям.

По отношению к информационным ресурсам сервис-ориентированной системы:

- автономные — обеспечивают выполнение операций трансформации и кодогенерации без использования модели онтологии и продукций (трансформация происходит без использования информационных ресурсов сервис-ориентированной системы);

- интегрируемые — обеспечивают выполнение операций с использованием информационных ресурсов сервис-ориентированной системы.

Интегрируемые программные компоненты имеют доступ к системным сервисам (средствам) визуального проектирования/разработки (редактирования) БЗ. Такими средствами являются два специализированных редактора: редактор визуального представления и редактирования модели онтологии и редактор визуального представления и редактирования продукционной модели в нотации RVML.

По типу выполняемых преобразований:

- интегрируемые компоненты анализа — обеспечивают преобразование концептуальных моделей в модель онтологии и продукций;
- интегрируемые компоненты кодогенерации — обеспечивают преобразование элементов модели онтологии в код БЗ на целевом ЯПЗ;
- автономные компоненты кодогенерации — обеспечивают преобразование концептуальных моделей в код БЗ на целевом ЯПЗ.

По характеру использования:

- внешние — фактически размещенные на других серверах, в программной системе непосредственно размещают только описание программных интерфейсов, для реализации используя структурный шаблон “адаптер”;
- размещенные непосредственно в составе сервис-ориентированной системы.

Таким образом, программный компонент (прикладной сервис) представляет собой простейший транслятор (конвертор), который переводит входные концептуальные модели, представленные на одном языке, в выходные БЗ на другом языке.

В соответствии с приведенной выше классификацией программных компонентов исходные и целевые метамодели могут быть представлены различными моделями:

$$MM_{IN} \in \{MM_{CM}, MM_{ONT}\}, \quad MM_{OUT} \in \{MM_{ONT}, MM_{KB}\},$$

где M_{ONT} — метамодель онтологической модели. Так, MM_{ONT} может выступать в качестве как исходной, так и целевой метамодели в процессе построения M_T. Описание MM_{ONT} представлено отдельно в рамках пункта описания модели расширенной онтологии.

При этом $CM \in \{M_{XML}, ONT\}$, $KB \in \{Code_{KRL}, ONT\}$, где ONT — модель онтологии.

Используя (5) и приведенную ранее классификацию программных компонентов, определим типы операторов преобразования:

$$T = \langle T_{CM-KB}, T_{CM-ONT}, T_{ONT-KB} \rangle, \tag{9}$$

где T_{CM-KB} — оператор преобразования исходной концептуальной модели в код БЗ на целевом ЯПЗ; T_{CM-ONT} — оператор преобразования исходной концептуальной модели в модель онтологии; T_{ONT-KB} — оператор преобразования модели онтологии в код БЗ на целевом ЯПЗ.

6.1. Архитектура типового программного компонента

Архитектура типового программного компонента, отражающая модель M_{TPC}, представлена на рис. 4.

Рис. 4. Архитектура типового программного компонента

6.2. Модель онтологии и продукций

Для унифицированного представления и хранения знаний, извлеченных из концептуальных моделей, предлагается использовать онтологическую модель. Данная модель позволяет абстрагироваться от особенностей описания знаний в различных ЯПЗ, используемых при реализации БЗ (например, CLIPS, JESS, Drools, RuleML, SWRL, OWL, RDF и др.), и хранить знания в собственном независимом формате. Полное описание моделей онтологии приводится в [26, 42].

Для обеспечения возможности работы со знаниями, представляемыми в форме правил, предлагается использовать специальную модель продукций. Особенностью данной модели является ее ориентация на использование специализированной графической нотации — Rule Visual Modeling Language (RVML) [6].

RVML — язык визуального моделирования правил, он предназначен для моделирования и описания логических правил, обладает большей выразительностью при описании причинно-следственных связей по сравнению с UML, в частности, RVML позволяет:

- использовать отдельные графические примитивы для отображения всех элементов продукций (а не стереотипы или типизированные классы, как в UML);
- присваивать отдельным фактам субъективные вероятности в виде коэффициентов уверенности;
- более наглядно отображать тип выполняемых действий (добавление, удаление, остановку);
- отображать логические операторы в условиях правил (“или” и “не”).

Данные модели являются информационным “ядром” сервис-ориентированной системы.

Заключение

Решение задач, связанных с разработкой новых методов и подходов к созданию ИС и их программных компонентов, остается перспективной областью научных исследований. В данной работе предложена технология автоматизированной разработки программных компонентов, предназначенных для формирования БЗ на основе трансформации концептуальных моделей.

В основе технологии использование: модели типового программного компонента, (включающей модель трансляции концептуальных моделей); предметно-ориентированного (декларативного) языка TMRL (Transformation Model Representation Language), предназначенного для представления и хранения модели трансляции; метода создания программных компонентов путем "клонирования" типового программного компонента и его настройки (специализации); SaaS-архитектуры (Software as a Service, программное обеспечение как услуга) сервис-ориентированного программного средства.

Технология позволяет создавать программные компоненты для извлечения знаний из различных концептуальных моделей и синтеза кода БЗ на целевом ЯПЗ. В качестве источников концептуальных моделей предлагается использовать группу моделей, для описания структуры которых используется XML (например, UML-модели, представленные в соответствии со стандартом XMI, или концепт-карты, представленные в соответствии со стандартом XTM). В качестве целевых ЯПЗ выбраны CLIPS и OWL.

В дальнейшем планируется подробнее исследовать возможность отчуждения и использования создаваемых программных компонентов в качестве автономных подсистем (сервисов) в составе сторонних ИС, в частности осуществить интеграцию программных компонентов с редактором продукционных баз знаний [43].

Благодарности. Работа выполнена при частичной финансовой поддержке РФФИ (гранты № 15-37-20655, 15-07-03088, 16-37-00122).

Список литературы / References

[1] **Гаврилова Т.А., Хорошевский В.Ф.** Базы знаний интеллектуальных систем. СПб.: Питер, 2000. 384 с.
Gavrilova, T.A., Khoroshevskiy, V.F. Knowledge bases of intelligent systems. SPb.: Piter, 2000. 384 p. (In Russ.)

[2] **Giarratano, J.C., Riley, G.D.** Expert SYStems: Principles and programming / 4th Edition. Thomson Course Technology, 2005. 288 p.

[3] **Рыбина Г.В.** Инструментальные средства построения динамических интегрированных экспертных систем: развитие комплекса АТ-ТЕХНОЛОГИЯ // Искусственный интеллект и принятие решений. 2010. № 1. С. 41–48.
Rybina, G.V. Instrumental tools for constructing of dynamic integrated expert system: developing of complex AT-TECHNOLOGY // Artificial Intelligence and Decision-Making. 2010. No. 1. P. 41–48. (In Russ.)

[4] **Загорулько Г.Б., Загорулько Ю.А.** Подход к организации комплексной поддержки процесса разработки интеллектуальных СППР в слабоформализованных предметных областях // Матер. VI Междунар. науч.-техн. конф. "Открытые семантические технологии проектирования интеллектуальных систем" (OSTIS-2016). Минск: БГУИР, 2016. С. 61–64.

Zagorulko, G.B., Zagorulko, Yu.A. An approach to organization of integrated support of the development of intelligent dss in weakly formalized domains // Proc. of the Intern. Conf. "Open Semantic Technologies for Intelligent Systems" (OSTIS-2016). Minsk: BGUIR, 2016. P. 61–64. (In Russ.)

[5] **Грибова В.В., Клещев А.С., Крылов Д.А., Москаленко Ф.М., Смагин С.В., Тимченко В.А., Тютюнник М.Б., Шалфеева Е.А.** Проект IACPaaS. Комплекс для интеллектуальных систем на основе облачных вычислений // Искусственный интеллект и принятие решений. 2011. № 1. С. 27–35.
Gribova, V.V., Kleshchev, A.S., Krylov, D.A., Moskalenko, F.M., Smagin, S.V., Timchenko, V.A., Tyutyunnik, M.B., Shalfeeva, E.A. Project IACPaaS. Complex for intelligent software based on cloud computing // Artificial Intelligence and Decision-making. 2011. No. 1. P. 27–35. (In Russ.)

[6] **Грищенко М.А., Юрин А.Ю., Павлов А.И.** Разработка экспертных систем на основе трансформации информационных моделей предметной области // Программные продукты и системы. 2013. № 3. С. 143–147.
Grishchenko, M.A., Yurin, A.Yu., Pavlov, A.I. Expert systems design based on the transformation of domain information models // Programmnye Produkty i Sistemy. 2013. No. 3. P. 143–147. (In Russ.)

[7] **Canadas, J., Palma, J., Tunez, S.** InSCo-Gen: A MDD tool for Web rule-based applications // LNCS. 2009. Vol. 5648. P. 523–526.

[8] **Djuric, D., Gasevic, D., Devedzic, V.** Ontology Modeling and MDA // J. of Object Technology. 2005. Vol. 4, No. 1. P. 109–128.

[9] **Парамонов В.В., Федоров Р.К., Бычков И.В., Черкашин Е.А., Ружников Г.М.** Технология синтеза каркаса информационной системы // Вычисл. технологии. 2010. Т. 15, № 6. С. 101–110.
Paramonov, V.V., Fedorov, R.K., Bychkov, I.V., Cherkashin, E.A., Ruzhnikov, G.M. Synthesis technology framework for an information system // Comput. Technologies. 2010. Vol. 15, No. 6. P. 101–110. (In Russ.)

[10] **Nalepa, G.J., Ligeza, A.** HeKatE methodology, hybrid engineering of intelligent systems // Intern. J. of Appl. Mathematics and Computer Sci. 2010. Vol. 20, No. 1. P. 35–53.

[11] **Kleppe, A., Warmer, J., Bast, W.** MDA explained: The model-driven architecture: Practice and promise / 1st Edition. Addison-Wesley Professional, 2003. 192 p.

[12] **Rozenberg, G.** Handbook of graph grammars and computing by graph transformations. World Scientific Publishing Company, 1997. 553 p.

[13] Object Management Group (OMG). Документация стандарта Query/View/Transformation (QVT), Version 1.2. Адрес доступа: http://www.omg.org/spec/QVT/1.2 (дата обращения: 10.11.2015).
Object Management Group (OMG). Query/View/Transformation (QVT), Version 1.2. Available at: http://www.omg.org/spec/QVT/1.2 (accessed 10.11.2015). (In Russ.)

[14] **Jouault, F., Allilaire, F., Bezivin, J., Kurtev, I.** ATL: A model transformation tool // Science of Computer Programming. 2008. Vol. 72, No. 1. P. 31–39.

[15] **Varro, D., Balogh, A.** The model transformation language of the VIATRA2 framework // Science of Computer Programming. 2007. Vol. 63, No. 3. P. 214–234.

[16] **Balasubramanian, D., Narayanan, A., Buskirk, C., Karsai, G.** The graph rewriting and transformation language: GReAT // Electronic Communications of the EASST. 2007. Vol. 1. P. 1–8.

[17] Рабочая группа W3C XSL. Документация стандарта XSL Transformations (XSLT), Version 2.0. Адрес доступа: http://www.w3.org/TR/xslt20 (дата обращения: 10.11.2015). W3C XSL Working Group. XSL Transformations (XSLT), Version 2.0. Available at: http://www.w3.org/TR/xslt20 (accessed 10.11.2015). (In Russ.)

[18] **Milanovic, M., Gasevic, D., Giurca, A., Wagner, G., Devedzic, V.** On interchanging between OWL/SWRL and UML/OCL // Proc. of 6th Workshop on OCL for (Meta-) Models in Multiple Application Domains (OCLApps) at the 9th ACM/IEEE Intern. Conf. on Model Driven Engineering Languages and Systems (MoDELS). Genoa, Italy, 2006. P. 81–95.

[19] **Zedlitz, J., Jorke, J., Luttenberger, N.** From UML to OWL 2 // Knowledge Technology. Berlin; Heidelberg: Springer, 2012. P. 154–163.

[20] **Felfernig, A., Friedrich, G.E., Jannach, D.** UML as domain specific language for the construction of knowledge-based configuration systems // Intern. J. of Software Engineering and Knowledge Engineering. 2000. Vol. 10, No. 4. P. 449–469.

[21] **Laera, L., Tamma, V., Bench-Capon, T., Semeraro, G.** SweetProlog: A system to integrate ontologies and rules // Rules and Rule Markup Languages for the Semantic Web. Berlin; Heidelberg: Springer, 2004. P. 188–193.

[22] **Gasevic, D., Djuric, D., Devedzic, V., Damjanovic, V.** Converting UML to OWL ontologies // Proc. of the 13th Intern. World Wide Web Conf. on Alternate Track Papers & Posters, 2004. P. 488–489.

[23] **Xu, Z., Ni, Y., He, W., Lin, L., Yan, Q.** Automatic extraction of OWL ontologies from UML class diagrams: a semantics-preserving approach // World Wide Web-Internet and Web Information Systems. 2012. Vol. 15, No. 5-6. P. 517–545.

[24] **Belghiat, A., Bourahla, M.** An approach based AToM3 for the generation of OWL ontologies from UML diagrams // Intern. J. of Computer Applications. 2012. Vol. 41, No. 3. P. 41–48.

[25] **Meditskos, G., Bassiliades, N.** CLIPS-OWL: A framework for providing object-oriented extensional ontology queries in a production rule engine // Data & Knowledge Engineering. 2011. Vol. 70, No. 7. P. 661–681.

[26] **Дородных Н.О., Юрин А.Ю.** Использование диаграмм классов UML для формирования продукционных баз знаний // Программная инженерия. 2015. № 4. С. 3–9.
Dorodnykh, N.O., Yurin, A.Yu. Using UML class diagrams for design of knowledge bases of rule-base expert systems // Software Engineering. 2015. No. 4. P. 3–9. (In Russ.)

[27] **Wang, E., Kim, Y.S.** A teaching strategies engine using translation from SWRL to Jess // 8th Intern. Conf. on Intelligent Tutoring Systems, ITS 2006. LNCS, 2006. Vol. 4053. P. 51–60.

[28] **Mei, J., Bontas, E.P., Lin, Z.** OWL2Jess: A transformational implementation of the OWL semantics // Proc. Parallel and Distributed Processing and Applications ISPA 2005 Workshops. 2005. P. 599–608.

[29] **Corsar, D., Sleeman, D.** Reusing JessTab rules in protege // Knowledge-Based Systems. 2006. Vol. 19, No. 5. P. 291–297.

[30] **Brilhante, V., Macedo, G.T., Macedo, S.F.** Heuristic transformation of well-constructed conceptual maps into OWL preliminary domain ontologies // Proc. Workshop on Ontologies and their Applications (WONTO 2006) at the 18th Brazilian Symposium on Artificial Intelligence. 2006.

[31] **Buyya, R., Yeoa, C.S., Venugopal, S., Broberg, J., Brandic, I.** Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility // Future Generation Computer Systems. 2009. Vol. 25, No. 6. P. 599–616.

[32] **Booch, G., Rumbaugh, J., Jacobson, I.** The unified modeling language user guide / 2nd Edition. New York: Addison-Wesley, 2005. 496 p.

[33] Object Management Group (OMG). Документация стандарта XML Metadata Interchange (XMI). Адрес доступа: http://www.omg.org/spec/XMI (дата обращения: 10.11.2015).
Object Management Group (OMG). XML Metadata Interchange (XMI). Available at: http://www.omg.org/spec/XMI (accessed 10.11.2015). (In Russ.)

[34] Рабочая группа TopicMaps.Org. Документация стандарта XML Topic Maps (XTM). Адрес доступа: http://www.topicmaps.org/xtm (дата обращения: 10.11.2015).
TopicMaps.Org Authoring Group. XML Topic Maps (XTM). Available at: http://www.topicmaps.org/xtm (accessed 10.11.2015). (In Russ.)

[35] **Частиков А.П., Гаврилова Т.А., Белов Д.Л.** Разработка экспертных систем. Среда CLIPS. СПб.: БХВ-Петербург, 2003. 608 с.
Chastikov, A.P., Gavrilova, T.A., Belov, D.L. Development of expert systems. CLIPS environment. SPb.: BKHV-Peterburg, 2003. 608 p. (In Russ.)

[36] **Grau, B.C., Horrocks, I., Motik, B., Parsia, B., Patel-Schneider, P., Sattler, U.** OWL 2: The next step for OWL // Web Semantics: Science, Services and Agents on the World Wide Web. 2008. Vol. 6, No. 4. P. 309–322.

[37] Software factories: assembling applications with patterns, models, frameworks, and tools / J. Greenfield, K. Short, S. Cook, S. Kent, J. Crupi. Wiley., 2004. 500 p.

[38] Компиляторы: принципы, технологии и инструментарий. 2-е изд.: Пер. с англ. / А.В. Ахо, М.С. Лам, Р. Сети, Дж.Д. Ульман. М.: Вильямс, 2008. 1184 с.
Compilers: Principles, techniques, and tools. 2nd ed. / A.V. Aho, M.S. Lam, R. Sethi, J.D. Ullman. Addison-Wesley, 2006. 1000 p.

[39] Рабочая группа W3C XML Schema. Документация стандарта XML Schema definition (XSD). Адрес доступа: http://www.w3.org/TR/xmlschema-0 (дата обращения: 10.11.2015).
W3C XML Schema Working Group. XML Schema definition (XSD). Available at: http://www.w3.org/TR/xmlschema-0 (accessed 10.11.2015). (In Russ.)

[40] **Khan, A., Sum, M.** Introducing design patterns in XML Schemas. Available at: http://www.oracle.com/technetwork/java/design-patterns-142138.html (accessed 10.11.2015).

[41] RVML Документация нотации Rule Visual Modeling Language (RVML). Адрес доступа: http://www.knowledge-core.ru/index.php?p=rvml (дата обращения: 10.11.2015).
RVML Rule Visual Modeling Language (RVML). Available at: http://www.knowledge-core.ru/index.php?p=rvml (accessed 10.11.2015). (In Russ.)

[42] **Николайчук О.А.** Методы, модели и инструментальное средство для исследования надежности и безопасности сложных технических систем: Автореф. д-ра техн. наук по спец. 05.13.01. М.: ИСА РАН, 2011. 37 с.
Nikolaychuk, O.A. Methods, models and tools for study of reliability and safety of complex technical systems: Abstract for degree of doctor of technical sciences in specialty 05.13.01. Moscow: ISA RAN, 2011. 37 p. (In Russ.)

[43] **Юрин А.Ю., Грищенко М.А.** Редактор баз знаний в формате CLIPS // Программные продукты и системы. 2012. № 4. С. 83–87.
Yurin, A.Yu., Grishchenko, M.A. Knowledge base editor for CLIPS // Programmnye Produkty i Sistemy. 2012. No. 4. P. 83–87. (In Russ.)

Поступила в редакцию 21 марта 2016 г.

Approach to the development of software components for generation of knowledge bases based on conceptual models

Bychkov, Igor V., Dorodnykh, Nikita O., Yurin, Alexander Yu.*

Matrosov Institute for System Dynamics and Control Theory of SB RAS, Irkutsk, 664033, Russia

*Corresponding author: Yurin, Alexander Yu., e-mail: iskander@icc.ru

The paper addresses the problem of improving the process of the design of intelligent systems and their components. The main problem in designing intelligent systems is a creation of knowledge bases. The efficiency of this process can be improved by the transformation of conceptual (information) models into program codes of knowledge bases. In turn, the conceptual models can be created with the aid of different CASE tools or software for cognitive and ontological modelling. Therefore, the creation of the unified approach (a technology) for developing software components for intelligent systems that provide generation of knowledge bases by the transformation of conceptual models is an actual problem. In this paper, the technology for the automated development of software components providing generation of knowledge bases as a result of the transformation of conceptual models is proposed. The main elements of technology are: the template of the software component for the models' translation; domain-specific (declarative) language for representation of translations — Transformation Model Representation Language (TMRL); method for development of software components based on emulation the template of the software component for the translation and specialization of a model; the conceptual architecture of the service-oriented software and its main elements, the software is based on the principles of the SaaS-model (Software as a Service). The proposed technology allows building of software components for knowledge extraction from various conceptual models as well as code generation for knowledge bases in the certain knowledge representation language. The conceptual models presented in XML are used as the source models (e. g., UML-models according to XMI, or concept maps according to XTM, etc.). CLIPS and OWL are selected as the target knowledge representation languages. The future work will focus on developing the autonomous software components (services) for the third-party intelligent systems.

Keywords: software components development, intelligent system, conceptual model, knowledge base, model transformation.

Acknowledgements. The reported study was partially supported by RFBR, research projects No. 15-37-20655, 15-07-03088, 16-37-00122.

Received 21 March 2016