4. **Boev V. D., Sypchenko R. P.** Komp'juternoe modelirovanie. Moscow, INTUIT.RU, 2010. 349 p. (in Russian).
5. **Emel'janov V. V., Jasinovskij S. I.** *Vvedenie v intellektual'noe imitacionnoe modelirovanie slozhnyh diskretnyh sistem i processov.* Jazyk RDO, Moscow, ANVIK, 1998, 427 p. (in Russian).
6. **Marka D., MakGoujen K.** *Metodologija strukturnogo analiza i proektirovanija SADT*, Moscow, MetaTehnologija, 1993, 240 p. (in Russian).
7. **Sorokin A. B.** Poliaspektnyj analiz pri proektirovanii sistem podderzhki prinjatija reshenij, *Nauchno-tehnicheskaja informacija. Serija 2. Informacionnye processy i sistemy*, 2014, no. 8, pp. 10—23 (in Russian).
8. **Teorija** sistem i sistemnyj analiz v upravlenii organizacijami: Spravochnik: Ucheb. posobie / Eds. V. N. Volkovoj, A. A. Emel'janova. Moscow, Finansy i statistika, 2006, 848 p. (in Russian).
9. **Forrester Dzh.** *Osnovy kibernetiki predprijatija*, Moscow, Progress, 1971, 340 p. (in Russian).
10. **Sorokin A. B., Bolotova L. S.** The evolutionary model as the projection methodology situationally — activity analysis and its realization on the example of the model against the development of infectious diseases, *Collection of scientific papers "Interactive systems: Problems of Human-Computer Interaction"*, Ulyanovsk, 2015. pp. 120—130.
11. **Bolotova L. S.** *Sistemy iskusstvennogo intellekta: modeli i tehnologii, osnovannye na znanijah*, Moscow, Finansy i statistika, 2012, 663 p. (in Russian).
12. **Sorokin A. B.** Svidetel'stvo o gosudarstvennoj registracii bazy dannyh № 2015621317 "Intellektual'naja baza dannyh dlja proektirovanija sistem osnovannyh na znanijah". Data gosudarstvennoj registracii v Reestre baz dannyh 26 avgusta 2015 goda.
13. **Shlee M.** *Qt 4.8. Professional'noe programmirovanie na C++*, Saint-Petersburg, BHV-Peterburg, 2012, 912 p. (in Russian).
14. **Sorokin A. B., Lobanov D. A.** Svidetel'stvo o gosudarstvennoj registracii programmy dlja JeVM № 2015662075 "Interpretator — processnyj plan". Data gosudarstvennoj registracii v Reestre programm dlja JeVM 17 nojabrja 2015 goda.

УДК 004.434

А. О. Сухов, канд. физ.-мат. наук, доц. каф., e-mail: ASuhov@hse.ru,
Е. Ю. Медведева, студент, e-mail: medvedevaeyu@mail.ru,
Национальный исследовательский университет "Высшая школа экономики", г. Пермь

Подход к разработке языкового инструментария для создания текстовых предметно-ориентированных языков

Описан подход к созданию языкового инструментария, новизна которого заключается в том, что в процессе функционирования языковой инструментарий выполняет интерпретацию моделей различных уровней иерархии, а не генерацию на их основе исходного кода. Это позволяет проводить настройку разработанных языков без перегенерации кода редактора, выполнять многоуровневое моделирование, а также определять правила преобразования написанных программ в код на целевом языке.

Ключевые слова: *предметно-ориентированные языки, текстовые языки, языковой инструментарий, метаязык, многоуровневое моделирование, метамодель, трансформация моделей, модельно-ориентированный подход*

Введение

Одной из движущих сил развития языков программирования и инструментальных средств разработки программного обеспечения является уменьшение трудовых затрат программистов за счет сокращения семантического разрыва между конструкциями языков программирования и объектами предметной области, которые требуется описать с помощью этих языков. Переход от машинных языков к языкам высокого уровня позволил существенно сократить трудозатраты и время разработки, а также повысить качество создаваемых систем. С начала 1970-х годов прошлого века стали активно развиваться проблемно- и предметно-ориентированные языки программирования, которые предназначены для решения определенного класса задач в конкретной предметной области. Эти языки позволили существенно сократить трудозатраты при разработке приложений, поскольку одной команде такого языка соответствуют десятки строк кода, написанных на языке высокого уровня [1—4]. Примерами предметно-ориентированных языков являются Prolog, VHDL, HTML, GPSS и др. С течением времени такие языки стали все чаще применяться при разработке программного обеспечения. На сегодняшний день разработка любой информационной системы немыслима без использования языка запросов SQL, языка проектирования интерфейса пользователя, интегрированного в IDE (Integrated Development Environment).

Предметно-ориентированные языки (Domain-Specific Languages, DSL) — языки программирования (моделирования), предназначенные для решения определенного круга задач в конкретной предметной области.

Поскольку такие языки оперируют терминами предметной области, при разработке и сопро-

вождении программных систем использовать их могут не только программисты, но и эксперты в предметной области, бизнес-аналитики, конечные пользователи. DSL могут иметь визуальный или текстовый синтаксис [5]. *Визуальные предметно-ориентированные языки*, как и любые графические схемы и модели, более наглядны, чем текстовые DSL. Это делает их понятными и выразительными: одна конструкция языка способна заменить несколько строк кода. *Текстовые предметно-ориентированные языки* позволяют программисту более точно и подробно выразить проектные решения, так как являются более формализованными.

При создании программных систем могут использоваться ранее разработанные предметно-ориентированные языки, однако не для всех предметных областей и задач подходят существующие решения, поэтому появляется необходимость создания новых текстовых DSL, что является довольно трудоемким процессом, поскольку помимо самого языка необходимо разработать текстовый редактор для работы с ним, генератор и отладчик кода.

Языковой инструментарий — это вид программного обеспечения, предназначенный для создания и сопровождения предметно-ориентированных языков. Он позволяет упростить процесс разработки DSL за счет автоматизированного построения среды разработки, которая включает редактор кода с подсветкой синтаксиса, браузер проектов, трансформатор для преобразования созданных на DSL программ в код на целевом языке программирования и отладчик кода.

Целью исследования является разработка подхода к созданию языкового инструментария, который позволит выполнять построение текстовых предметно-ориентированных языков, написание программ с помощью этих языков и их преобразование в код на целевом языке.

Языковой инструментарий должен быть прост в использовании и понятен различным категориям пользователей, поскольку участие представителей заказчика является неотъемлемой частью современного процесса разработки программного обеспечения.

Анализ языковых инструментариев

В процессе разработки и сопровождения программного обеспечения часто возникает необходимость внесения изменений в ранее созданное решение. Причины этих изменений могут быть различны: в процессе анализа или проектирования были допущены неточности, произошло изменение предметной области, заказчик модифицировал требования к разрабатываемой системе и др. Все это может привести к необходимости внесения изменений в описание предметно-ориентированного языка, которые повлекут за собой соответствующие модификации в редакторе языка. Для того чтобы существовала возможность внесения изменений в DSL без перегенерации исходного кода редактора, необходимо, чтобы языковой инструментарий в процессе своего функционирования проводил интерпретацию описания языка, а не генерацию на его основе исходного кода анализаторов, как это делает большинство генераторов лексических и синтаксических анализаторов Lex, YACC, ANTRL и др.

Для описания синтаксиса DSL в языковых инструментариях используется метаязык. Большинство инструментариев предоставляет пользователю заранее предопределенный неизменяемый набор конструкций, "вшитый" в реализацию, что накладывает ряд ограничений на возможность гибкой настройки языкового инструментария и разрабатываемых DSL на предметную область. Возможность изменения описания метаязыка сделает языковой инструментарий более гибким и позволит пользователю самостоятельно настраивать метаязык, применяемый для построения предметно-ориентированных языков. Кроме того, такая возможность позволит выполнять многоуровневое моделирование. Определив предметно-ориентированный язык, можно загрузить его в языковой инструментарий и использовать в качестве метаязыка. Это предоставит возможность создавать предметно-ориентированные метаязыки, что еще в большей степени адаптирует систему к особенностям конкретной предметной области.

Для преобразования написанных с помощью DSL программ в код на целевом языке программирования необходимо предоставить пользователю удобные средства описания такой трансформации. Данные средства должны позволять определять правила преобразования конструкций DSL в целевой язык, поскольку для каждого DSL и целевого языка эти правила будут уникальны.

При всем при этом языковой инструментарий не должен быть перегружен лишней функциональностью, которая не востребована при разработке и эксплуатации предметно-ориентированных языков, так как это может затруднить процесс использования языкового инструментария пользователями, которые не являются профессиональными IT-специалистами.

Существуют различные языковые инструментарии для создания текстовых DSL: OpenArchitectureWare, Meta Programming System, IDE Meta-Tooling Platform, MontiCore, Spoofax Language Workbench и др. Выполним сравнение наиболее развитых из них по следующим критериям:

- возможность создания DSL для широкого спектра предметных областей;
- возможность многоуровневого моделирования;
- возможность изменения описания DSL без перегенерации кода его редактора;

- возможность определения правил трансформации конструкций DSL в целевой язык;
- наличие избыточной функциональности языкового инструментария, не используемой при создании DSL.

Eclipse OpenArchitectureWare (oAW) — это инструментарий с открытым исходным кодом для реализации модельно-ориентированной разработки приложений, основанный на платформе Eclipse [6].

Для описания синтаксиса разрабатываемых DSL используется компонент xText [7], поэтому возможность изменения метаязыка и многоуровневого моделирования отсутствует. На основе построенной грамматики выполняется генерация лексического и синтаксического анализаторов языка. При каждом изменении описания языка требуется повторная генерация кода его редактора.

Для описания трансформаций создаваемых программ в целевое представление используется язык xPand. Генерация кода проводится только на языке Java.

Система *Meta Programming System* (MPS) разработана компанией JetBrains и используется совместно со средой разработки Java-приложений IntelliJ-IDEA [8, 9].

Для описания DSL данный языковой инструментарий предоставляет в распоряжение пользователя отдельные языки: язык структуры, язык редактора, базовый язык, язык шаблонов, каждый из которых применяется для определения отдельного аспекта DSL.

Абстрактный синтаксис языка описывается с помощью встроенного метаязыка, ключевым элементом которого является концепт. Концепт в MPS — это тип узла абстрактного синтаксического дерева, содержащий информацию о свойствах, методах, потомках и отношениях узла. Возможность многоуровневого моделирования в инструментарии отсутствует.

Язык редактора позволяет описать конкретный синтаксис DSL. Далее с помощью языка шаблонов следует определить правила преобразования конструкций создаваемого языка в код на языке Java. После этого требуется выполнить генерацию кода редактора языка. Для внесения изменений в язык необходимо повторно выполнить все этапы описания языка и перегенерировать код редактора, т. е. возможность динамического изменения описания языка отсутствует.

Языковой инструментарий создан на базе среды разработки IntelliJ-IDEA, которая предназначена для решения более широкого круга задач, чем создание DSL, поэтому языковой инструментарий обладает излишней функциональностью, не используемой в процессе разработки DSL.

IDE Meta-Tooling Platform (IMP) — это плагин к среде Eclipse, разработанный компанией IBM в целях упрощения процесса создания IDE для новых языков программирования [10].

При использовании данного плагина необходимо определить только конкретный синтаксис языка, представляющий собой контекстно-свободную грамматику, а абстрактный синтаксис строится автоматически с помощью генератора синтаксического анализатора LPG. Причем изменить средства описания конкретного синтаксиса невозможно; как следствие, отсутствует возможность многоуровневого моделирования.

Описание правил трансформации моделей в исходный код на целевом языке выполняется на языке Java. После внесения изменений в описание языка следует выполнить повторную генерацию кода редактора DSL.

Поскольку данный языковой инструментарий является плагином к среде Eclipse, то его использование затруднено дополнительной функциональностью данной IDE.

MontiCore (MC) — языковой инструментарий, базирующийся на платформе Eclipse [11—13].

Данный программный продукт для определения DSL использует метаязык для описания конкретного и абстрактного синтаксиса, который подобен входному формату системы ANTLR. Возможности изменения описания метаязыка и многоуровневого моделирования отсутствуют.

По окончании описания языка необходимо выполнить компиляцию проекта, после чего будет запущен новый экземпляр Eclipse, в котором в качестве инструмента будет выступать разработанный DSL, поэтому возможность внесения изменений в описание предметно-ориентированного языка без перегенерации исходного кода редактора отсутствует.

На основе построенной с помощью DSL модели может быть сгенерирован код на языке Java, изменение правил генерации кода невозможно.

Инструментарий MC является плагином к среде Eclipse, поэтому он перегружен широкими возможностями данной IDE.

Spoofax Language Workbench (SLW) — языковой инструментарий, который позволяет итеративно разрабатывать текстовые предметно-ориентированные языки и выполнять запуск написанных с их помощью программ в экземпляре среды Eclipse [14—16].

SLW предлагает пользователю среду разработки DSL, включающую в себя компоненты описания и анализа синтаксиса, трансформации и генерации объектного кода.

Для описания синтаксиса языка SLW использует собственный метаязык SDF3, который объединяет конкретный и абстрактный синтаксисы в едином шаблоне.

Для выполнения преобразований моделей в текстовое представление на целевом языке применятся язык Stratego, который позволяет определить текстовые шаблоны для генерации кода.

Результаты сравнения языковых инструментариев

Критерий сравнения	oAW	MPS	IMP	MC	SLW
Возможность создания DSL для широкого спектра предметных областей	+	+	+	+	+
Возможность многоуровневого моделирования	—	—	—	—	—
Возможность изменения описания DSL без перегенерации кода его редактора	—	—	—	—	—
Возможность определения правил трансформации	+	+	+	—	+
Наличие избыточной функциональности	+	+	+	+	+

К ограничениям системы следует отнести отсутствие возможности многоуровневого моделирования и перегруженность среды разработки Eclipse функциями, которые не используются при создании DSL.

В таблице приведены сводные сравнительные характеристики рассмотренных языковых инструментариев.

Подводя итог, можно говорить о том, что все рассмотренные языковые инструментарии являются лишь надстройкой над более мощной средой программирования (Eclipse или IntelliJ-IDEA). Это приводит к тому, что данные инструментарии сложны для освоения из-за наличия избыточной функциональности. Кроме того, "отчуждение" созданных DSL от среды разработки для их дальнейшего использования при разработке и сопровождении программного обеспечения становится невозможным.

Ни один из языковых инструментариев не предоставляет возможности многоуровневого моделирования, что усложняет их настройку на специфику предметной области. Также ни один инструментарий не позволяет динамически изменять описание языка без повторной генерации исходного кода редактора; это затрудняет процесс модификации языка на этапе эксплуатации, так как для этих целей требуется иметь IDE.

Большинство рассмотренных языковых инструментариев позволяют выполнять преобразование созданных моделей только в код на языке Java, что также существенно ограничивает возможность их применения.

Существующие проблемы описания и использования текстовых предметно-ориентированных языков, ограничения языковых инструментариев и подходов к созданию DSL было решено устранить при разработке языкового инструментария, удовлетворяющего следующим требованиям:

- универсальность — возможность построения языков для широкого спектра предметных областей;
- наличие возможности многоуровневого моделирования;
- наличие возможности внесения изменений в описание языков без перегенерации кода редакторов DSL;
- единообразие средств представления, описания и использования моделей различных уровней иерархии: построение моделей и работа с ними должны проводиться унифицированно с помощью одних и тех же программных средств;
- наличие возможности задания правил трансформации программ, написанных с помощью DSL, в код на целевом языке;
- удобство работы с системой для различных категорий пользователей.

Построение текстовых DSL

На основе требований, предъявляемых к языковому инструментарию, разработаем подход к созданию текстовых предметно-ориентированных языков.

Процесс создания DSL начинается с построения его *метамодели* (задания абстрактного синтаксиса), для описания которой используется *метаязык*. Именно благодаря наличию метаязыка языковой инструментарий позволяет разрабатывать DSL для широкого спектра предметных областей.

После построения метамодели пользователь получает в распоряжение язык, с помощью которого может выполнять разработку программы — описание *модели предметной области*, содержащей основные объекты, связи между ними и операции над объектами.

Во время исполнения программы на основе модели предметной области строится *модель состояния*, объекты которой — экземпляры элементов модели предметной области и связей между ними.

Таким образом, во время разработки DSL создается целая иерархия моделей, представленная на рис. 1.

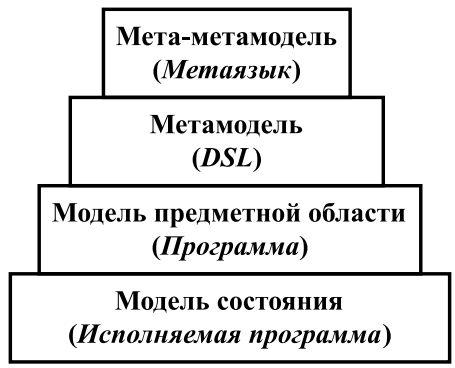

Рис. 1. Иерархия моделей при разработке DSL

Построение иерархии моделей и унифицированная обработка моделей различных уровней иерархии позволяют выполнять многоуровневое моделирование, которое предоставляет возможность в большой степени настроить разрабатываемый язык на специфику конкретной предметной области. Предположим, что необходимо разработать текстовый предметно-ориентированный язык для описания обслуживания на автозаправочной станции. Для этого в качестве метаязыка могут быть использованы формы Бэкуса — Наура (БНФ). Однако применять такой метаязык смогут только IT-специалисты. Если требуется предоставить конечному пользователю возможность самостоятельно определять, изменять DSL, то необходимо, чтобы метаязык был ему понятен. Для этого с помощью БНФ может быть разработан текстовый предметно-ориентированный язык для систем массового обслуживания, далее он должен быть загружен в языковой инструментарий и использован в качестве предметно-ориентированного метаязыка при создании DSL для описания работы автозаправочной станции.

По окончании создания программы (модели предметной области) необходимо, чтобы ее код, написанный на DSL, был преобразован в код на целевом языке. Правила трансформации будут задаваться с помощью текстовых шаблонов, содержащих статическую и динамическую части. Статическая часть правила трансформации не зависит от модели и является неизменной. Динамическая часть содержит изменяемые фрагменты кода, которые зависят от значения атрибутов элементов модели.

Метаязык языкового инструментария

Метаязык языкового инструментария должен быть лаконичным, понятным различным категориям пользователей, а также позволять описывать метамодели для различных предметных областей, не ограничивая уровня детализации.

Для описания синтаксиса языка, с одной стороны, можно использовать стандартные метаязыки, например, расширенные формы Бэкуса — Наура (РБНФ), а с другой стороны, можно сделать систему более гибкой и настраиваемой за счет предоставления пользователю возможности самостоятельно модифицировать конструкции метаязыка. Однако в любом случае следует задать набор базовых терминальных метасимволов и обозначить правила построения метамоделей.

За основу формального описания синтаксиса текстовых DSL были взяты РБНФ. При разработке языка в описание его синтаксиса автоматически включаются предопределенные нетерминалы "буква", "цифра", "идентификатор", которые используются практически в любой предметной области.

Для возможности изменения встроенного метаязыка необходимо, чтобы языковой инструментарий в процессе своего функционирования выполнял интерпретацию созданных моделей, а не генерацию на их основе программного редактора.

Архитектура языкового инструментария

В процесс разработки DSL могут быть вовлечены различные категории пользователей (рис. 2). IT-специалисты совместно с бизнес-аналитиками выполняют настройку метаязыка на предметную область (в случае необходимости), создают текстовые языки, определяют правила преобразования конструкций языка в код на целевом языке. Бизнес-аналитики совместно с конечными пользователями выполняют построение моделей с использованием разработанного DSL и генерацию кода на целевом языке.

Для построения текстовых предметно-ориентированных языков необходимо предоставить пользователю удобную среду разработки, в которой он мог бы создавать новые языки, строить с их помощью модели и генерировать код на целевом языке. Архитектура языкового инструментария представлена на рис. 3.

Единым хранилищем информации обо всех проектах является *репозиторий*. Он хранит как отдельные метамодели (предметно-ориентированные языки), так и готовые решения (DSL и созданные на его основе программы). Также вместе с языком хранятся правила трансформации программ в целевые языки.

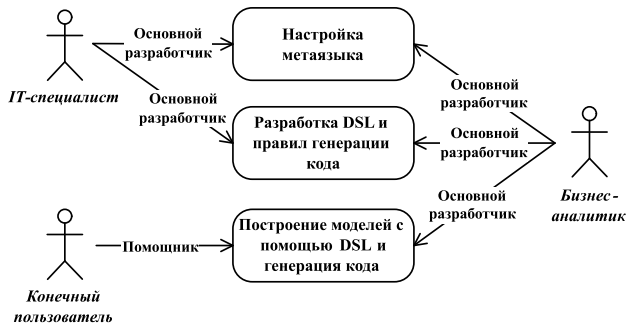

Рис. 2. Пользователи языкового инструментария

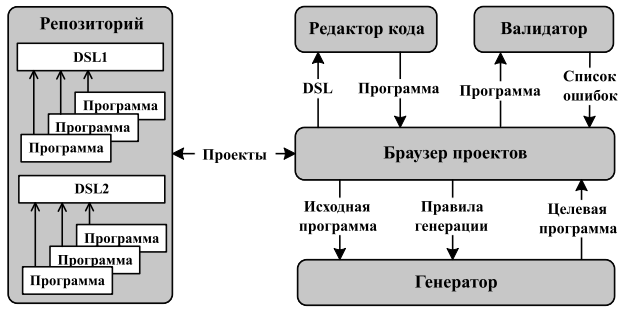

Рис. 3. Архитектура языкового инструментария

Браузер проектов предоставляет пользователю возможность загружать из репозитория ранее сохраненные проекты, добавлять, удалять, редактировать метамодели, созданные на их основе модели, определять правила трансформации моделей.

Редактирование моделей различных уровней иерархии осуществляется в *текстовом редакторе кода*, который выполняет подсветку синтаксиса, автодополнение конструкций языка в соответствии с заданными правилами, а для метамодели — еще и проверку корректности описания языка (все ли нетерминалы описаны, нет ли повторного описания нетерминала и др.).

Создание программы на основе построенного DSL контролируется *валидатором*, который преобразует метамодель в дерево синтаксического разбора и выполняет проверку модели на лексическую и синтаксическую корректность.

Преобразование созданной модели предметной области в код на целевом языке программирования осуществляется *генератором*, который на основе заданных в браузере проектов правил трансформации и исходной программы выполняет построение кода на целевом языке.

Благодаря тому, что хранение и обработка моделей различных уровней иерархии выполняются единообразно, появляется возможность выполнять многоуровневое моделирование, проводить объединение нескольких моделей, созданных на основе общей метамодели.

Заключение

Благодаря тому, что языковой инструментарий в процессе своего функционирования выполняет интерпретацию описания моделей различных уровней иерархии, существует возможность внесения изменений в описание DSL во время работы системы, без перегенерации исходного кода редактора языка. Это делает языковой инструментарий более гибким и настраиваемым на изменившиеся условия эксплуатации без необходимости привлечения профессиональных IT-специалистов.

Унификация обработки моделей и метамоделей предоставляет возможность выполнять мультиязыковое программирование на основе объединения нескольких метамоделей в одну.

Возможность изменения описания метаязыка и многоуровневого моделирования позволяет приблизить метаязык к специфике предметной области.

Предоставление пользователю возможности определять правила трансформации позволяет выполнять генерацию кода на различных целевых языках, что также делает систему более гибкой.

Благодаря тому, что языковой инструментарий будет разрабатываться "с нуля", а не как надстройка над существующей IDE, он не будет загроможден излишней функциональностью, которая не используется в процессе разработки DSL. Это сделает систему понятной и простой в применении для различных категорий пользователей.

Теоретическая значимость работы заключается в том, что описанный подход к разработке языкового инструментария для создания текстовых динамически настраиваемых предметно-ориентированных языков позволяет устранить недостатки большинства существующих инструментов.

Практическая значимость подхода заключается в том, что он позволяет разработать языковой инструментарий для создания DSL, построения программ с использованием этих языков и генерации кода на целевом языке на основе определенных пользователем правил трансформации.

Список литературы

1. **Кирсанова А. А.** Проблематика использования текстовых DSL в информационных системах // Вестник ЮУрГУ. Сер. "Компьютерные технологии, управление, радиоэлектроника". 2015. Т. 15, № 3. С. 5—10.
2. **Globalizing** Domain-Specific Languages // Lecture Notes in Computer Science. Berlin: Springer. 2015. Vol. 9400. 88 p.
3. **Drobintsev P. D.** Model-Oriented Approach for Industrial Software Development // Modeling and Analysis of Information Systems. 2015. Vol. 22, № 6. P. 750—762.
4. **Rumpe B.** Agile Development with Domain Specific Languages // Modelling Foundations and Applications. Lecture Notes in Computer Science. Berlin: Springer. 2011. Vol. 6698. P. 387—388.
5. **Сухов А. О.** Классификация предметно-ориентированных языков и языковых инструментариев // Математика программных систем: межвузовский сборник научных статей. Пермь: Изд-во Пермского государственного национального исследовательского университета. 2012. Вып. 9. С. 74—83.
6. **Haase A.** Introduction to OpenArchitectureWare 4.1.2 // Model-Driven Development Tool Implementers Forum 2007. URL: https://pdfs.semanticscholar.org/a6fa/d330ed634810f3f0026 58443ab361eb3c423.pdf (дата обращения: 05.06.2017).
7. **Жуков М. В.** Реализация предметно-ориентированных языков средствами фреймворка Xtext // Изв. Пензенского государственного педагогического университета имени В. Г. Белинского. Сер. "Физико-математические и технические науки". 2009. № 13. С. 98—102.
8. **Pech V., Shatalin A., Voelter M.** JetBrains MPS as a Tool for Extending Java // Proceedings of the 2013 International Conference on Principles and Practices of Programming on the Java Platform: Virtual Machines, Languages, and Tools. NY. 2013. P. 165—168.
9. **Voelter M.** Lessons Learned from Developing Mbeddr: a Case Study in Language Engineering with MPS // Software & Systems Modeling. 2017. P. 1—46.
10. **Charles Ph., Fuhrer R. M., Sutton S. M.** IMP: a Meta-Tooling Platform for Creating Language-Specific IDEs in Eclipse // Proceedings of the Twenty-Second IEEE/ACM International Conference on Automated Software Engineering. NY. 2007. P. 485—488.
11. **Krahn H., Rumpe B., Völkel S.** MontiCore: a Framework for Compositional Development of Domain Specific Languages // International Journal on Software Tools for Technology Transfer. 2010. Vol. 12. Iss. 5. P. 353—372.
12. **Krahn H., Rumpe B., Völkel S.** MontiCore: Modular Development of Textual Domain Specific Languages // Objects, Components, Models and Patterns. Lecture Notes in Business Information Processing. Berlin: Springer. 2008. Vol. 11. P. 297—315.
13. **Heim R.** Compositional Language Engineering Using Generated, Extensible, Static Type-Safe Visitors // Modelling Foundations and Applications. Lecture Notes in Computer Science. Berlin: Springer. 2016. Vol. 9764. P. 67—82.

14. **Kats L. C. L., Visser E.** The Spoofax Language Workbench: Rules for Declarative Specification of Languages and IDEs // Proceedings of the Conference "Object-Oriented Programming, Systems, Languages and Applications" (OOPSLA 2010). 2010. P. 444—463.

15. **Vos B., Kats L. C. L., Pronk C.** EpiSpin: An Eclipse Plug-In for Promela/Spin Using Spoofax // Model Checking Software. Lecture Notes in Computer Science. Berlin: Springer. 2011. Vol. 6823. P. 177—182.

16. **Wachsmuth G. H., Konat G. D., Visser E.** Language Design with the Spoofax Language Workbench // IEEE Software. 2014. Vol. 31. P. 35—43.

A. O. Sukhov, Associate Professor, e-mail: ASuhov@hse.ru,
E. Yu. Medvedeva, Student, e-mail: medvedevaeyu@mail.ru,
National Research University Higher School of Economics, Perm, 614070, Russian Federation

An Approach to Language Workbench Development for Creating Textual Domain-Specific Languages

The usage of domain-specific languages at the software development process can reduce labor costs, because one command of this language corresponds to dozens of code lines written in a high-level language. However, at DSL creation it is also necessary to develop a convenient environment for working with it that includes a text editor, code generator and debugger. Language workbench is software designed to create and maintain domain-specific languages.

The paper describes an approach to the development of language workbench for creating of textual dynamically customizable domain-specific languages. The approach novelty is that in the process of functioning the language workbench interprets the models of different levels of hierarchy, rather than generates on their basis of the editor source code. This allows to configure the developed language for changed operating conditions without re-generation of the editor code, to perform multi-level modeling, and to determine the rules for conversion of written programs into code in the target language.

As the metalanguage for DSL construction, it is proposed to use the extended Backus-Naur forms. The architecture of the language workbench includes a repository, text code editor, project browser, validator, and generator.

Different categories of users can be involved at the DSL development process: IT-specialists, business analysts, end-users, etc.

Keywords: *domain-specific languages, textual languages, language workbench, architecture, metalanguage, multi-level modeling, metamodel, model transformation, model-oriented approach, language tuning*

References

1. **Kirsanova A. A.** Problematika ispolzovanija tekstovyh DSL v informacionnyh sistemakh (Problems of Using Textual DSL in Information Systems), *Vestnik JuUrGU. Serija "Kompjuternye tehnologii, upravlenie, radiojelektronika"*, 2015, vol. 15, no. 3, pp. 5—10 (in Russian).

2. **Globalizing** Domain-Specific Languages, *Lecture Notes in Computer Science*. Berlin: Springer, 2015, vol. 9400, 88 p.

3. **Drobintsev P. D.** Model-Oriented Approach for Industrial Software Development, *Modeling and Analysis of Information Systems*, 2015, vol. 22, no 6, pp. 750—762.

4. **Rumpe B.** Agile Development with Domain Specific Languages, *Modelling Foundations and Applications. Lecture Notes in Computer Science*. Berlin: Springer, 2011, vol. 6698, pp. 387—388.

5. **Suhov A. O.** Klassifikacija predmetno-orientirovannyh jazykov i jazykovykh instrumentariev (Domain-specific Languages and Language Workbench Classification), *Matematika programmnyh sistem: mezhvuzovskij sbornik nauchnyh statej*, Perm: Izdatelstvo Permskogo gosudarstvennogo nacionalnogo issledovatelskogo universiteta, 2012, no. 9, pp. 74—83 (in Russian).

6. **Haase A.** Introduction to OpenArchitectureWare 4.1.2, Model-Driven Development Tool Implementers Forum — 2007. URL: https://pdfs.semanticscholar.org/a6fa/d330ed634810f-3f002658443ab361eb3c423.pdf (Access at: 05.06.2017).

7. **Zhukov M. V.** Realizacija predmetno-orientirovannykh jazykov sredstvami frejmvorka Xtext (The Realization of Object-oriented Languages by Means of Framework Xtext), *Izvestija Penzenskogo gosudarstvennogo pedagogicheskogo universiteta imeni V. G. Belinskogo. Serija "Fiziko-matematicheskie i tehnicheskie nauki"*, 2009, no. 13, pp. 98—102 (in Russian).

8. **Pech V., Shatalin A., Voelter M.** JetBrains MPS as a Tool for Extending Java, *Proceedings of the 2013 International Conference on Principles and Practices of Programming on the Java Platform: Virtual Machines, Languages, and Tools*. NY, 2013, pp. 165—168.

9. **Voelter M.** Lessons Learned from Developing Mbeddr: a Case Study in Language Engineering with MPS, *Software & Systems Modeling*, 2017, pp. 1—46.

10. **Charles Ph., Fuhrer R. M., Sutton S. M.** IMP: a Meta-Tooling Platform for Creating Language-Specific IDEs in Eclipse, *Proceedings of the Twenty-Second IEEE/ACM International Conference on Automated Software Engineering*. NY, 2007, pp. 485—488.

11. **Krahn H., Rumpe B., Völkel S.** MontiCore: a Framework for Compositional Development of Domain Specific Languages, *International Journal on Software Tools for Technology Transfer*, 2010, vol. 12, iss. 5, pp. 353—372.

12. **Krahn H., Rumpe B., Völkel S.** MontiCore: Modular Development of Textual Domain Specific Languages, *Objects, Components, Models and Patterns. Lecture Notes in Business Information Processing*. Berlin: Springer, 2008, vol. 11, pp. 297—315.

13. **Heim R.** Compositional Language Engineering Using Generated, Extensible, Static Type-Safe Visitors, *Modelling Foundations and Applications. Lecture Notes in Computer Science*. Berlin: Springer, 2016, vol. 9764, pp. 67—82.

14. **Kats L. C. L., Visser E.** The Spoofax Language Workbench: Rules for Declarative Specification of Languages and IDEs, *Proceedings of the Conference "Object-Oriented Programming, Systems, Languages and Applications" (OOPSLA 2010)*. Nevada, 2010, pp. 444—463.

15. **Vos B., Kats L. C. L., Pronk C.** EpiSpin: An Eclipse Plug-In for Promela/Spin Using Spoofax, *Model Checking Software. Lecture Notes in Computer Science*. Berlin: Springer, 2011, vol. 6823, pp. 177—182.

16. **Wachsmuth G. H., Konat G. D., Visser E.** Language Design with the Spoofax Language Workbench, *IEEE Software*, 2014, vol. 31, pp. 35—43.