НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

НАУЧНЫЙ ДОКЛАД

по результатам подготовленной
научно-квалификационной работы (диссертации)
Построение и анализ моделей трансформации предметно-ориентированных языков в условиях динамических контекстов

ФИО Улитин Борис Игоревич

Направление подготовки 09.06.01 Информатика и вычислительная техника

Профиль (направленность) программы Математическое моделирование, численные методы и комплексы программ

Аспирантская школа по компьютерным наукам

Аспирант _____/Улитин Б.И. /
подпись

Научный руководитель _____/Бабкин Э.А. /
подпись

Директор
Аспирантской школы по компьютерным наукам____ _____/Объедков С.А. /
подпись

Нижний Новгород, 2020

СОДЕРЖАНИЕ

АКТУАЛЬНОСТЬ, ЦЕЛЬ И ЗАДАЧИ ИССЛЕДОВАНИЯ

В настоящее время широкое распространение получают предметно-ориентированные языки (ПОЯ), причем как среди исследователей, так и среди практиков [27, 32]. В первую очередь это связано с тем, что ПОЯ представляют собой удобный, понятный и достаточно простой с точки зрения пользователя механизм управления той предметной областью, для которой они созданы [19].

Исследованиям проблем разработки ПОЯ посвящены работы как инотранных ученых, в частности, Pablo Gómez-Abajo, Esther Guerra, Juan de Lara [20], Aleksandar Popovic, Ivan Lukovic, Vladimir Dimitrieski, Verislav Djuki [37], Walter Cazzola, EdoardoVacchi, так и отечественных, в частности, И.С. Ануреева [1], Б.Н. Гайфуллина и В.Е. Туманова [2], А.О. Сухова [4], В.Г. Федоренкова и П.В. Балакшина [5] и др.

При анализе вопроса разработки и использования ПОЯ, в первую очередь исследователи учитывают тот факт, что ПОЯ в своей основе должен соответствовать той предметной области, для которой создается. Все исследователи ([13, 14, 19, 32]) отмечают тот факт, что в основе любого ПОЯ лежит некоторая модель, которая является зеркальным отражением той предметной области, в рамках которой и создается ПОЯ. Фактически, модель определяет не только структуру ПОЯ, но также и его семантическое, смысловое наполнение. Определяет его поведение и механизмы работы с ним. Так, например, Cleenewerck отмечает, что эффективность ПОЯ целиком и полностью зависит от полноты его внутренней модели [13]. Также исследователи сходятся во мнении, что ПОЯ с момента своего создания могут эволюционировать, изменяться. Эволюция может происходить как под влиянием изменений самой предметной области ([13, 14]), так и под влиянием внутренних факторов, таких как изменение поведения пользователей системы [29], их гетерогенность [36].

С другой стороны, в работах не освещен сам механизм отслеживания изменений предметной области и их трансляции на модель ПОЯ. Более того, считается, что к моменту разработки ПОЯ сама модель предметной области уже

создана и неким образом перенесена в модель ПОЯ (как, например, в работах [20, 24]). Справедливо отметить, что ряд исследователей, в их числе Peter Bell [1], Josh G.M. Mengerink, Alexander Serebrenik, Mark van den Brand [31], Ramon R.H.Schiffelers [30] и в частности, Jonathan Sprinkle, Gabor Karsai [39], исследуют вопрос эволюции графических моделей предметной области. Но они не рассматривают последующий перенос проведенных изменений моделей и правил, их обеспечивающих, на модели ПОЯ [39]. Напротив, они стараются сохранить структуру ПОЯ неизменной, что не соответствует предположению, что модель ПОЯ идентична модели предметной области, а значит, любое изменение модели предметной области должно выливаться в эквивалентное изменение модели ПОЯ.

Кроме того, при работе с ПОЯ встает вопрос о компетенциях тех пользователей, которые его используют. Ведь чем богаче, шире тот набор компетенций и опыта, которым обладает пользователь ПОЯ, тем более сложные конструкции он может использовать и тем богаче средства языка, которые ему необходимы. Каждый такой пользователь, в зависимости от собственного набора компетенций, в неявном виде задает собственную модель использования ПОЯ, а значит, и собственную его модификацию. Тем самым мы получаем динамический контекст использования ПОЯ, зависящий и создаваемый набором компетенций непосредственного пользователя ПОЯ. К сожалению, данный вопрос не рассматривается в работах исследователей. В них чаще всего считается, что модель ПОЯ соответствует лишь модели предметной области [20] и может меняться лишь под влиянием изменений в ней [25]. Пользователь же во время работы имеет доступ исключительно к конструкциям той версии языка, которая ему дана изначально, не может комбинировать их, образуя новые, собственные языковые лексемы и конструкции, тем самым меняя структуру и, как следствие, модель самого ПОЯ.

Таким образом, мы получаем, фактически, два направления эволюции ПОЯ: эволюция, вызванная изменения модели предметной области [3], и эволюция, вызванная изменением наборов компетенций пользователя ПОЯ. В таком случае возникает вопрос, каким образом оценить, какая из явных и искусственно созданных моделей ПОЯ является наиболее эффективной и каков общий тренд

развития каждой из линий жизни ПОЯ. Необходимы некоторые формальные критерии, позволяющие оценить эффективность каждой из построенных моделей ПОЯ. В качестве таких критериев могут быть выделены критерии, предложенные в работе Parastoo Mohagheghi and Øystein Haugen «Evaluating Domain-Specific Modelling Solutions» [34]. Они предлагают оценивать эффективность ПОЯ с помощью интегрального показателя, основанного на трех группах критериев: синтаксическая полнота, семантическая полнота и прагматичность ПОЯ [34]. Синтаксическая полнота отвечает, фактически, за соответствие ПОЯ своей предметной области. Семантическая – за то, насколько полно система ограничений ПОЯ соответствует ограничения предметной области. Прагматичность же отвечает за понимание пользователем конструкций ПОЯ и возможность их адаптации и комбинации под собственные требования. К сожалению, исследователи не рассматривают подробно сам процесс расчета данных критериев, не указывают, каким именно образом можно провести оценку полноты соответствия ПОЯ своей предметной области.

Исходя из вышесказанного, возникает потребность в разработке формальных моделей, позволяющих перенести процесс эволюции моделей предметной области и моделей компетенций пользователей ПОЯ на эволюцию модели самого ПОЯ.

На основании вышеизложенного, ***целью исследования*** является разработка методов моделирования эволюции ПОЯ и их последующий анализ с применением систем компьютерного моделирования.

Для достижения данной цели необходимо реализовать следующие ***задачи исследования***:

- Провести критический анализ методов анализа соответствия и непротиворечивости графовых моделей,
- Разработать формальную модель, лежащую в основе предлагаемого подхода к моделированию трансформации и эволюции моделей ПОЯ,
- Разработать алгоритмы трансформации моделей (как предметной области, так и ПОЯ),

- Разработать прототип программного комплекса трансформации моделей на основе разработанной формальной модели и алгоритмов,
- Провести апробацию и тестирование полученных результатов (моделей, алгоритмов и программных прототипов) на ПОЯ для различных предметных областей.

Объектом исследования являются формальные модели, позволяющие описать структуру ПОЯ, модель его использования разными пользователями и желаемые способы трансформации ПОЯ с целью повышения эффективности его использования.

Предметом исследования являются процессы трансформации ПОЯ в динамическом контексте в случае изменения предметной области или компетенций пользователей ПОЯ.

СТЕПЕНЬ РАЗРАБОТАННОСТИ ТЕМЫ ИССЛЕДОВАНИЯ

В настоящее время информационные системы представляют собой все более сложные программные среды. Если раньше предполагалось, что каждая отдельная программа исполняет одну или несколько функций [18], то сейчас информационные системы являются более сервисно-ориентированными в том смысле, что состоят из множества взаимосвязанных модулей, каждый из которых отвечает за свою функциональность [13, 17].

Как следствие, каждое изменение в информационную систему в целом может привести к необходимости каскадного изменения множества ее отдельных частей и связей между ними [13]. Тем более важно обеспечить возможность проведения независимой модификации каждого отдельного компонента информационной системы. Это также важно в той точки зрения, что отдельные компоненты информационной системы могут использоваться различными категориями пользователей, ответственными за те или иные задачи.

Именно поэтому в настоящее время жизненный цикл информационной системы включает в себя не только этапы проектирования и разработки, но также дальнейшей поддержки и эволюции информационной системы [19].

Последние два этапа являются наименее изученными, поскольку чаще всего являются зависимыми от языка реализации информационной системы, количества конечных пользователей и количества модулей в информационной системе и пр. [19, 29]

Более того, разделение информационной системы на блоки визуальные и функциональные приводит к еще большей сложности внесения изменений в нее в автоматизированном режиме [13]. Это связано в первую очередь с тем, что отдельные конечные пользователи обладают различным опытом использования информационных систем, имеют собственные предпочтения и компетенции. Как следствие, они могут использовать информационную систему по-разному, требуя ее модификации под собственные требования. Как итог получается набор

идентичных по функционалу, но различных по визуальному отображению информационных систем [41].

Для того, чтобы упростить данную модель изменения информационной системы под требования пользователей мы считаем целесообразным рассматривать визуальную составляющую информационной системы в виде предметно-ориентированного языка. Это эффективно и обоснованно с той точки зрения, что графический интерфейс информационной системы обладает ограниченной функциональностью (непосредственно связанной с функционалом), как и ПОЯ, и может в этом смысле рассматриваться как особый вид ПОЯ. Именно поэтому внедрение элементов эволюции ПОЯ в эволюцию информационных систем может быть эффективным и обеспечить большую гибкость последних в отношении требований различных категорий пользователей [14].

Предметно-ориентированный язык (ПОЯ) является языком программирования, созданным для некоторой предметной области [19]. С этой точки зрения ПОЯ формализует внутри себя структуру, поведение, а также ограничения и требования данной предметной области. В отличие от языков общего назначения, не имеющих строгой привязки к предметной области, ПОЯ обладают ограниченной выразительностью и обеспечивают поддержку только верхнего уровня абстракции предметной области [8]. Как следствие, ПОЯ, фактически, содержит внутри себя некоторую модель предметной области, тем самым позволяя пользователям оперировать непосредственными терминами и концептами предметной области. Более того, на уровне ПОЯ также сохраняется возможность учитывать ограничения предметной области, исключая некорректные выражения [13].

С точки зрения структуры ПОЯ может рассматриваться как в лингвистическом, так и в математическом виде. С лингвистической точки зрения ПОЯ является хоть и искусственно созданным, но языком предметной области и содержит в себе, как и любой язык, семантическую и синтаксическую составляющие [22]. С другой стороны, ПОЯ является языком программирования,

основанным на концептах предметной области и обладающим ограниченной выразительностью с точки зрения набора операций, доступных пользователям. С одной стороны, синтаксис ПОЯ содержит все концепты, термины, ограничения и пр. компоненты предметной области. В то же время отдельные компоненты ПОЯ могут отличаться от присутствующих в предметной области, поскольку являются некоторого рода абстракциями над прототипами оригинальных компонентов предметной области [21]. В этом смысле синтаксис ПОЯ может не строго следовать синтаксическим правилам, существующим в предметной области и, как следствие, определять свой конкретный синтаксис над абстрактной моделью предметной области, взятой за основу при его разработке. Подводя итог данным рассуждениям, можно обобщить, что структура ПОЯ содержит в себе семантику и синтаксис, который, в свою очередь, разделяется на абстрактный и конкретный. Семантика языка полностью определяется концептуальной (семантической) моделью предметной области [25], наделяя синтаксические компоненты смыслом, гарантируя, таким образом, когерентность ПОЯ в целом с предметной областью. Синтаксические же уровни ПОЯ определяют непосредственно представление семантических составляющих в терминах конкретных команд и выражений ПОЯ. Абстрактный синтаксис определяет набор объектов, которые присутствуют и могут быть использованы в языке, тем самым являясь некоторой проекцией семантической составляющей языка [24]. Конкретный же синтаксис в данном случае выражает фактические обозначения для тех или иных объектов абстрактного синтаксиса, определяет способы их задания в терминах конкретного языка. Такая структура синтаксиса ПОЯ полностью соответствует предположению, что для одной предметной области может быть созданы несколько ПОЯ и различные диалекты одного и того же ПОЯ могут использовать одну и ту же абстрактную модель предметной области [15], определяя лишь различные команды конкретного синтаксиса для работы с ней.

Однако, хоть данная трехуровневая структура ПОЯ и рассматривается в литературе, на практике чаще всего работа проводится только с уровнем

синтаксиса языка. Большинство современных исследователей сходятся во мнении, что чаще всего в настоящее время процесс разработки ПОЯ начинается с синтаксического уровня, либо полностью опуская, либо лишь упоминая на уровне концепции семантическую составляющую языка. В частности, такой позиции придерживаются такие исследователи как Ruffolo M., Sidhu I., Guadagno L. [38], Challenger M., Demirkol S., Getir S., Mernik M., Kardas G., Kosar T. [11], которые в своих работах придерживаются классического цикла создания ПОЯ, описанного подробно в книге Fowler M. [19]. Можно утверждать, что такой подход к разработке ПОЯ также связан с особенностями существующих инструментов разработки ПОЯ, в частности, редакторов языка, таких как Xtext [16], в которых ПОЯ описывается с помощью грамматических инструментов, тем самым определяя его синтаксис без необходимости определения семантической составляющей.

Такое ограничение существующих инструментов становится существенным, если учесть возможность наличия несоответствий между семантической составляющей языка и синтаксическими конструкциями, вызванными ограничениями инструментов разработки языка (например, ограничения кардинальности в отношениях между сущностями не всегда могут быть описаны в грамматике – пример таких противоречий приведен в работах Haav et al. [23] и Kosar et al. [28]). Однако, если мы описываем только синтаксис ПОЯ, то найти такие противоречия можем только в процессе его эксплуатации, вручную, что, опять же, ведет к необходимости внесения исправлений в ПОЯ.

Более того, реализация ПОЯ, начиная с его синтаксической части, ведет нас к невозможности организации согласованной и непрерывной эволюции ПОЯ, поскольку каждая новая версия языка не связана с предыдущей, а существует независимо от нее (поскольку каждая из низ основана на собственной грамматике). В предположении, что ПОЯ создан для конкретной предметной области и что любая предметная область имеет тенденцию к эволюции с течением времени, такой подход делает использование ПОЯ крайне затруднительным, поскольку требует

постоянного присутствия специалиста-разработчика для внесения необходимых корректив в язык.

В то же время, поскольку ПОЯ основан на предметной области, то на семантическом уровне он содержит все те концепты, которые в ней присутствуют [21]. Как следствие, если мы можем формализовать модель предметной области, мы можем в дальнейшем сопоставить ее с моделью семантики ПОЯ и вынести решение о соответствии ПОЯ оригинальной предметной области. Более того, такая формальная модель предметной области может быть использована и на уровне абстрактного синтаксиса языка, поскольку уже содержит в себе описание всех необходимых объектов языка.

Абстрактный синтаксис языка обычно описывается в виде метамодели, включающей в себя основные концепты предметной области (языка), отношения между ними, а также набор правил, специфицирующих ограничения как на концепты, так и на отношения между ними в рамках предметной области.

В этом смысле абстрактный синтаксис (метамодель) ПОЯ представляется в виде артефакта, содержащего информацию об объектах языка и связях между ними без конкретизации поведенческих аспектов. Такое объектно-ориентированное описание метамодели ПОЯ позволяет свести процедуру проверки соответствии ПОЯ предметной области к проверке соответствия метамодели ПОЯ с концептуальной моделью предметной области, позволяя организовать ее автоматизацию.

Однако, как было сказано ранее, существующие инструменты реализации ПОЯ, за исключением MetaEdit+ [33], поддерживают описание ПОЯ в грамматической форме. Тем самым, устраняя все возможные преимущества определения метамодели ПОЯ в объектно-ориентированном виде. Более того, такой способ определения ПОЯ полностью устраняет возможность автоматизации проверки соответствия ПОЯ с концептуальной моделью предметной области. В этом смысле MetaEdit+ отличается от остальных инструментов, поскольку позволяет описывать именно объектную структуру языка, которую в дальнейшем

можно использовать для непосредственного описания предметной области и ее концептов. С другой стороны, MetaEdit+ является графическим инструментом разработки ПОЯ, тем самым не позволяя определить поведение ПОЯ через конкретный синтаксис. В этом смысле можно утверждать, что метамодель ПОЯ в существующих инструментах явно не выражается, что является их существенным недостатком как для разработчиков (необходимо проводить ручную проверку соответствия ПОЯ предметной области), так и для конечных пользователей, которые не могут вносить изменения в ПОЯ без знания языка спецификации выбранного инструмента.

Более эффективным в данном случае видится возможность реализации метамодели ПОЯ в объектно-ориентированном виде, поскольку ее содержание очень похоже на концептуальную модель предметной области (множество концептов и отношения между ними) и может быть получено путем преобразования последней, однако данный подход реализован в настоящее время только в одном инструменте разработки ПОЯ MetaEdit+, который, однако, имеет ряд других ограничений, о которых было сказано ранее.

В большинстве современных исследований [19, 27, 28, 35] процесс разработки ПОЯ включает в себя следующие этапы: принятие решения (о необходимости создания ПОЯ, иначе говоря, анализ его применимости), анализ предметной области (для которой создается ПОЯ), проектирование ПОЯ (включающее в себя определение всей структуры ПОЯ и выбор наиболее подходящего вида ПОЯ), реализация ПОЯ и развёртывание ПОЯ. Также иногда рассматривают как отдельный этап поддержку ПОЯ (включающую в себя возможность эволюции ПОЯ) [13].

Заметим, что в данном случае не имеет значения, какой вид ПОЯ разрабатывается, текстовый или графический (визуальный), поскольку различия между данными видами ПОЯ становятся существенными только на этапе непосредственной реализации ПОЯ. Более детально различия между двумя видами ПОЯ можно найти в работе Сухова А.О. [4].

В нашем случае наиболее существенными для рассмотрения являются этапы проектирования и реализации ПОЯ, а также этап эволюции, который целиком зависит от того, каким образом был реализован ПОЯ.

Обычно деятельность по проектированию и реализации ПОЯ включает в себя формализацию абстрактного синтаксиса создаваемого языка (чаще всего в виде метамодели), определение конкретного синтаксиса, автоматическую генерацию по данным описаниям редактора языка, описание (в том или ином виде) семантики языка. Как правило, в этом случае задается денотационная семантика путем создания генератора из моделей на визуальном языка в какой-либо текстовый язык. Однако, возможно использование и задание правил интерпретации ПОЯ (например, в виде операционной семантики в виде набора правил преобразования моделей).

Отдельный вопрос, который необходимо упомянуть и который зачастую игнорируется в работах по реализации визуальных ПОЯ, но часто упоминается в статьях по реализации текстовых – вопрос переиспользования синтаксиса уже созданных ПОЯ. На первый взгляд кажется, что данный вопрос актуален только для текстовых языков, однако, как показано в [29], данный вопрос актуален для обоих видов ПОЯ, поскольку при реализации любого типа ПОЯ можно использовать уже существующие языки как базу для создаваемого, путём создания различных расширений (например, с помощью механизма профилей UML), либо путём переиспользования и расширения частей существующих метамоделей.

Однако, существующие технологии в большинстве своем не поддерживают данную тенденцию и вынуждают пользователя создавать ПОЯ с нуля (исключение составляют MetaEdit+ [33] и XTend [16], однако они ограничены либо вовсе не имеют функционала по созданию конкретного синтаксиса ПОЯ и потому также реализуют данную особенность в полной мере). Для того же, чтобы поддерживать полное переиспользование метамоделей, в инструментах необходимо внедрить специальные средства, позволяющие организовать трансформацию различных метамоделей с возможность последующей их декомпозиции и/или импорта. В этом

смысле может являться полезным обобщенное модельно-ориентированное описание ПОЯ, поскольку оно позволит создать единую структуру переходов между моделями различных уровней абстракции и оригинальной метамоделью, созданной на основе концептуальной модели предметной области.

Как следствие данных ограничений существующих сред, эволюция ПОЯ в них является либо труднореализуемой, либо не реализуемой вовсе, поскольку каждое изменение в ПОЯ приводит к необходимости пересоздания его структуры целиком.

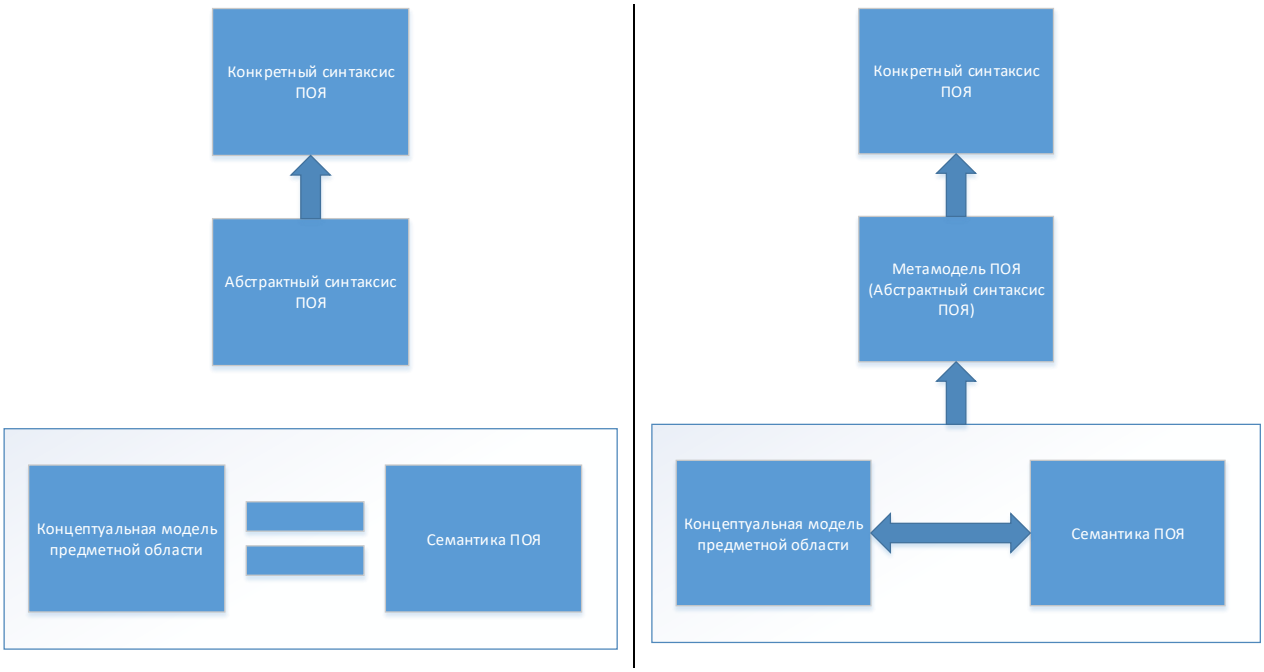

Рис. 1: Классический и предлагаемый подходы разработки ПОЯ

Таким образом, на основании вышеизложенной структуры ПОЯ, а также основных стадий жизненного цикла ПОЯ мы можем изложить основные положения классической методологии разработки ПОЯ.

Мы опускаем этапы принятия решения о разработке ПОЯ и его непосредственного внедрения, поскольку они не связаны непосредственно с разработкой ПОЯ в том смысле, что принятие решения сказывается лишь на том, будет ли в принципе разработан ПОЯ или его более удобный программный аналог, а этап внедрения связан уже с готовым ПОЯ и не зависит от того, в каком виде он реализован.

В классической методологии, как мы уже показали ранее, предполагается,

что анализ предметной области при водит нас к тому, что мы определяем основные объекты предметной области и связи между ними, тем самым формируя семантику будущего ПОЯ. Однако в формальном виде данная модель предметной области нигде не фиксируется и не используется на этапе проектирования синтаксиса ПОЯ. Как следствие, мы начинаем разработку ПОЯ непосредственно с уровня синтаксиса, тем самым делая невозможным любое утверждение о согласованности различных версий ПОЯ между собой, так как сравнение должно происходить на уровне семантики, а не синтаксиса. Кроме того, в зависимости от типа ПОЯ мы начинаем работу либо с уровня абстрактного синтаксиса (визуальные ПОЯ), не разрабатывая конкретный синтаксис (его функциональные компоненты), либо с грамматического описания конкретного синтаксиса (текстовые ПОЯ). Тем самым, мы, фактически, разрабатываем всегда новый ПОЯ, не оставляя возможности для модификации ранее созданных версий, поскольку каждый раз необходимо модифицировать синтаксис без возможности установления соответствия его различных компонентов в новой и прежней версиях.

Такая структура и последовательность разработки ПОЯ нарушает всю логику работы над ним в соответствии с жизненным циклом ПОЯ. В частности, мы каждый раз работаем с синтаксисом ПОЯ и при необходимости внесения изменений из-за эволюции предметной области вынуждены заново разрабатывать новый ПОЯ с нуля, а не использовать результаты, полученные на этапе анализа предметной области в виде концептуальной модели.

Более того, любое изменение в ПОЯ может быть внесено только специалистами, обладающими соответствующими навыками программирования, поскольку конечные пользователи не могут напрямую менять синтаксис языка, описанные в терминах грамматики, не связанной с их предметной областью. В итоге складывается ситуация, когда ПОЯ с течением времени становится неактуальным и теряет свое основное предназначение – строгое соответствие предметной области.

В литературе были предприняты определенные попытки устранения данных ограничений классической методологии разработки ПОЯ. Так, например, в работах

Bell [8] приводятся примеры реализации семейства диалектов ПОЯ на основании единой грамматической структуры. Однако, данные подход требует реализации единой и полной грамматической структуры ПОЯ, не решая проблему ее дальнейшей модификации при изменении модели предметной области.

Таким образом, существует потребность разработать иной подход к разработке ПОЯ, который позволял бы не только формально представлять результаты каждого отдельного этапа жизненного цикла ПОЯ, но также использовать их на последующих. Для этого предлагается использовать модельно-ориентированный подход, который позволит представить каждый компонент ПОЯ в виде единообразной объектно-ориентированной модели, позволяя вносить изменения в различные компоненты независимо.

В данном случае любое изменение ПОЯ может быть сформулировано посредством трансформаций различных моделей между собой, начиная от концептуальной модели предметной области, заканчивая объектно-ориентированным представлением конкретного синтаксиса языка. Это позволяет не только выработать единый подход к разработке ПОЯ, но также в дальнейшем организовать непрерывную его эволюцию: любое изменение в концептуальной модели предметной области может быть перенесено посредством преобразования в мета-модель ПОЯ, которая, в свою очередь, переносится в конкретную модель диалекта языка. Главным преимуществом такой схемы является согласованность новых диалектов языка с предыдущими на уровне метамодели, что позволяет настраивать ПОЯ в реальном режиме времени конечными пользователями без необходимости пересоздания всей структуры ПОЯ с нуля.

ОСНОВНЫЕ РЕЗУЛЬТАТЫ ИССЛЕДОВАНИЯ И ПОЛОЖЕНИЯ, ВЫНОСИМЫЕ НА ЗАЩИТУ

Основными результатами данного исследования являются следующие:

- полная объектно-ориентированная формализация структуры ПОЯ;
- описание механизма непрерывной разработки и эволюции ПОЯ с помощью кросс-модельных преобразований (проекций) на основании семантической модели предметной области;
- механизм автоматизации разработки интерфейсов как особого вида ПОЯ через инструменты инвариантов и кросс-модельных преобразований.

Как было показано ранее, при разработке ПОЯ в классическом подходе разработка ПОЯ включается в себя работу только над синтаксисом, либо абстрактным, либо сразу конкретным, полностью опуская этап работы с семантикой языка. Более того, при разработке структуры синтаксиса ПОЯ полностью игнорируется тот факт, что ПОЯ основан на концептах предметной области, а значит, содержит внутри себя некоторую модель предметной области, представляющую основные объекты и связи между ними (вместе с ограничениями). Такое ограничение связано, как было сказано ранее, с особенностями сред разработки ПОЯ.

Для устранения данного недостатка существующих сред мы предлагаем разрабатывать структуру ПОЯ (его метамодель) как проекцию (результат кросс-модельной трансформации [6]) семантической модели предметной области, которая может быть формализована в виде [7]:

$$DSM = (\mathcal{H}_C, \mathcal{H}_R, O, R, A, M, D) \quad (1)$$

где

- $\mathcal{H}_C$ и $\mathcal{H}_R$ представляют собой множества классов и отношений. Классы содержат набор атрибутов, тип каждого из которых также является классом. В обоих множествах $\mathcal{H}_C$ и $\mathcal{H}_R$ определены частичные порядки, позволяющие представлять концепты и таксономии отношений соответственно;

- O и R представляют собой множества классов и отношений мжду ними, также называемых объектами и кортежами соответственно;
- A представляет собой множество аксиом, описанных специальными правилами, выражающими ограничения предметной области;
- M представляет собой набор модулей вывода, которые представляют собой логические программы, состоящие из набора (дизъюнктивных) правил, которые позволяют рассуждать о представленных и сохраненных знаниях, позволяя выводить новые, не объявленные явно ранее, знания;
- D представляет собой набор дескрипторов (т. е. правил вывода в двумерной объектно-ориентированной грамматике атрибутов), позволяющий распознавать экземпляры класса (концепции), содержащиеся в O, а также их аннотации.

На основании данной модели мы разрабатываем структуру ПОЯ, которая также может быть формализована в объектно-ориентированной форме, причем, на всех уровнях.

Уровень семантики, который наполняет смыслом конструкции ПОЯ, является полностью тождественным концептуальной модели предметной области. Тем самым, утверждается его модельно-ориентированная форма представления.

Абстрактный синтаксис ПОЯ (метамодель) может быть получен путем проекции множества объектов целевой предметной области и операций над ними [41]. Учитывая, что любая модель представляет собой комбинацию (E, R) некоторого множества сущностей и отношений между ними, метамодель ПОЯ может быть формализована в модельно-ориентированном виде в следующей форме [40]:

- ***Множество сущностей метамодели*** $E = \{e_i\}$, $i \in \mathbb{N}$, $i < \infty$, где каждая сущность $e_i = \{SName_i, SICount_i, Attr_i, Opp_i, SRest_i\}$ характеризуется своим названием ($SName_i$, уникальное в рамках конкретной модели), допустимым количеством экземпляров сущности ($SICount_i \in \mathbb{N}$, $SICount_i \geq 0$), множеством атрибутов ($Attr_i = \{attr_{j_i}\}$, $j_i \in \mathbb{N}$, $j_i < \infty$), множеством

операций над экземплярами сущности ($Opp_i = \{opp_{j_i}\},\ j_i \in \mathbb{N},\ j_i < \infty$) и множеством and ограничений ($SRest_i = \{srest_{j_i}\},\ j_i \in \mathbb{N},\ j_i < \infty$).

- ***Множество отношений между сущностями*** $R = \{r_i\},\ i \in \mathbb{N},\ i < \infty$, где каждое отношение $r_i = \{RName_i, RType_i, RMult_i, RRest_i\}$ определяется своим названием ($RName_i$, уникальное в рамках конкретной модели), типом ($RType_i \in \mathbb{N},\ RType_i \geq 0$), определяемым природой отношения, множественностью ($RMult_i \in \mathbb{N},\ RMult_i \geq 0$), которая определяет, сколько экземпляров сущностей, участвующих в отношении, могут быть использованы, и множеством ограничений ($RRest_i = \{rrest_{j_i}\},\ j_i \in \mathbb{N},\ j_i < \infty$).

Основываясь на данных положениях, структура метамодели ПОЯ может быть определена как $(E, R, Rest, Opp)$, где первые два элемента E and Rel представляют собой объектный уровень, а остальные $Rest = \bigcup_{i=1}^{|E|} SRest_i \bigcup_{i=1}^{|E|} RRest_i$, Opp представляют собой функциональные аспекты работы с ПОЯ. Интерпретируя множество E как множество сущностей предметной области, R как множество отношений между ними и $Rest, Opp$ как множество операций над сущностями и ограничения на них, можно утверждать, что структура семантической модели предметной области (СМПО) и структура метамодели ПОЯ соответствуют друг другу. Как следствие, можно организовать автоматизированную трансформацию их между собой посредством кросс-модельных преобразований.

В случае конкретного синтаксиса, мы утверждаем, что объекты на уровне конкретного синтаксиса являются проекциями объектов уровня семантического [41]. Следовательно, в этом случае мы получаем полную эквивалентность уровней семантики и синтаксиса ПОЯ.

Функциональные же аспекты на синтаксическом уровне представляют собой множество операций, которые позволяют определить контекст создаваемых объектов и их поведение.

Как следствие, синтаксический уровень ПОЯ может быть формализован в

виде тройки $(O_{syntax}, R_{syntax}, Rule_{syntax})$, где $O_{syntax} \subseteq E$ и $R_{syntax} \subseteq R$ являются подмножествами множеств объектов и отношений между ними метамодели ПОЯ, а $Rule_{syntax}$ представляет собой множество правил, описывающих отображения между метамоделью и конкретным синтаксисом ПОЯ [40].

Самое главное здесь то, что такое определение конкретного синтаксиса ПОЯ на основе его метамодели не зависит от типа конкретного синтаксиса ПОЯ (например, текстовый или визуальный). Для визуальных языков необходимо установить связи между сущностями и визуальными символами, которые их представляют - как это сделано, например, с GMF. Точно так же с текстовыми языками, которые требуют связи между элементами метамодели и синтаксическими структурами текстового ПОЯ. Примером такого подхода является TCS.

В этих условиях мы можем сказать о полном модельно-ориентированном представлении структуры синтаксиса ПОЯ. Структура позволяет не только описывать оба уровня синтаксиса ПОЯ структурированным и унифицированным образом, но и оптимизировать процесс разработки и дальнейшего развития ПОЯ путем введения нескольких синтаксических диалектов ПОЯ на одной неизменяемой метамодели. Более того, версификация ПОЯ может быть обеспечена аналогичным образом как на мета-уровне, так и на уровне конкретного синтаксиса, без необходимости воссоздавать всю структуру ПОЯ каждый раз, когда требуются изменения. Это важно, поскольку ПОЯ может иметь несколько конкретных синтаксисов, объединенных единой метамоделью.

Учитывая все вышесказанное, мы можем утверждать, что структура ПОЯ может быть полностью формализована в виде объектно-ориентированной модели, а значит, получена путем кросс-модельных трансформаций (M2M-преобразования [9]) из семантической модели предметной области [41]. В соответствии с данным утверждением, процесс разработки ПОЯ принимает следующий иерархический вид (рис. 2).

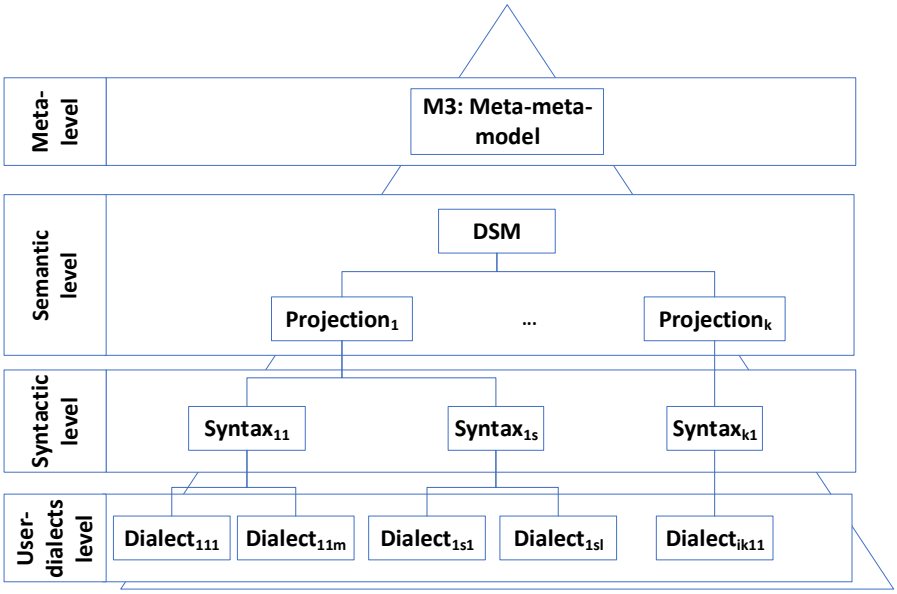

Рис. 2: Семантическая иерархия проектирования ПОЯ на основе проекций

В нашем случае иерархия разделена на четыре уровня в соответствии с этапами разработки ПОЯ. Каждый следующий (нижний) уровень основан на модельных артефактах предшествующего (верхнего) уровня. Единая М3 мета-метамодель определяет общие основания для всех мета-моделей и нижних уровней.

Структура семантической иерархии определяет соответствующий процесс создания ПОЯ [41]. Он начинается с определения семантической модели предметной области (СМПО), содержащей все ключевые объекты целевой предметной области и отношения между ними. Когда СМПО создана, мы можем построить семантическую модель ПОЯ с помощью операции семантической проекции. Любая семантическая проекция выполняет некое M2M-преобразование СМПО в какой-то его фрагмент. Таким образом, семантическая проекция полностью определяет семантическую модель того или иного диалекта ПОЯ. В этом случае семантическая модель становится объектно-временной структурой, поскольку ее следует адаптировать в соответствии с изменениями в СМПО с течением времени, тем самым определяя новое заполнение объекта СМПО.

После того, как выполнена семантическая проекция, синтаксический уровень ПОЯ может быть получен путем M2M преобразования результата соответствующей проекции [10]. Что важно, получаемые синтаксические модели ПОЯ независимы друг от друга и определяются конечными пользователями в

соответствии с адаптацией семантической проекции к их собственным задачам. Наконец, созданные синтаксисы используются конечными пользователями ПОЯ, которые определяют набор диалектов ПОЯ в рамках одной конкретной синтаксической модели.

Как следствие, ПОЯ создается с учетом знаний о предметной области и требований пользователей. Первый факт дает возможность организовать согласованные изменения в СМПО и семантической модели ПОЯ, а второй гарантирует возможность принять синтаксис ПОЯ без необходимости переопределять семантический уровень ПОЯ.

На рис. 3 показаны отличия традиционных подходов от предлагаемого нами проекционного. Традиционные подходы начинаются с «ручного» определения конкретного синтаксиса ПОЯ, за которым следует перевод синтаксиса в терминах грамматики. Следовательно, каждое изменение в целевой предметной области приводит к необходимости переопределения конкретного синтаксиса ПОЯ и пересоздания соответствующей грамматики. Аналогичный процесс повторяется в случае, когда изменения в ПОЯ вызваны конечными пользователями. В результате традиционные подходы на выходе получают несовместимые диалекты ПОЯ, преобразование и переходы между которыми являются недоступными из-за различий на всех уровнях структуры ПОЯ.

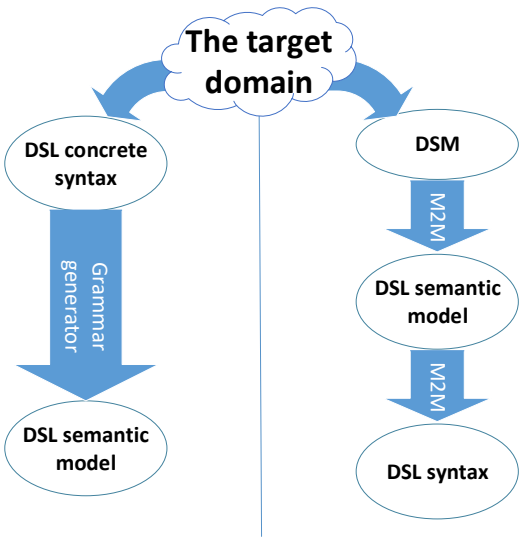

Рис. 3: Схема различий между традиционным (слева) и проекционным (справа) подходами к разработке ПОЯ

По сравнению с традиционными подходами, предлагаемая проекционная схема (рис. 3 слева) разработки ПОЯ организована в строгом соответствии с целевой предметной областью. Такое соответствие обеспечивается за счет применения последовательных проекций между различными моделями полуавтоматическим способом посредством преобразований M2M: из СМПО в семантическую модель ПОЯ, а затем в синтаксическую модель конкретного диалекта ПОЯ. В результате мы можем определить несколько синтаксических диалектов ПОЯ на основе одной конкретной семантической модели ПОЯ [41]. Причем данные диалекты будут согласованы и преобразуемы друг к другу без переопределения семантических моделей ПОЯ [42].

Наконец, учитывая, что чаще всего разработка ПОЯ предполагает также разработку некоторой программной среды и программного интерфейса, мы можем обобщить наш подход и на их разработку [43]. В данном случае мы основываемся на том факте, что интерфейс, как и ПОЯ, содержит в своей основе множество объектов и операции над ними, а значит, может быть основан на объектах, заложенных в основе структуры ПОЯ. С этой точки зрения объектно-ориентированное представление интерфейса может быть получено также путем проекции соответствующих объектов метамодели ПОЯ в объекты интерфейса [44].

Однако, в данном случае структура объектов метамодели ПОЯ и интерфейса имеют различную декомпозицию: объекты интерфейса являются суперпозицией отдельных мини-шаблонов, из которых строятся объекты интерфейса, а объекты метамодели – своих атрибутов. Из-за такого расхождения в природе структуры интерфейсов и метамодели требуется введение дополнительных понятий для установления соответствия между компонентами интерфейса и метамодели и гарантии корректности проводимых проекций.

Таким понятием является понятие инварианта [44]. С точки зрения наиболее общего подхода, инвариант – это свойство, принадлежащее классу объектов, которое остается неизменным, когда к объектам применяются преобразования определенного типа [12]. С этой точки зрения инвариант можно интерпретировать двумя способами [26]: (i) набор объектов, которые остаются неизменными во время

предусмотренного преобразования, (ii) операция, которая может применяться к нескольким объектам одновременно (например, операция ПЕРЕИМЕНОВАНИЕ, которые меняют имя объекта независимо от его типа). Принимая во внимание эти идеи, инварианты разделяются на два класса: структурные и функциональные (индуктивные и операционные) инварианты. В обоих случаях инвариант определяется при некотором преобразовании (переходе) множества объектов.

Индуктивный инвариант может быть определен непосредственно в терминах транзитивной системы и называется транзитивным (или индуктивным) и обозначается inv_τ [12]. Для этого типа инвариантов определены два типа спецификаций, $next$ и inv:

$$next_\tau.(p,q).F \triangleq [p \Longrightarrow \mathcal{W}.F.q], \tag{2}$$

$$inv_\tau.p.F \triangleq next_\tau.(p,p).F \wedge [\mathcal{J}.F \Longrightarrow p] \tag{3}$$

Неформально, $next_\tau.(p,q)$ значит, что в любом случае, когда имеет место переход из состояния, удовлетворяющего p, результирующее состояние будет удовлетворять условию q. Аналогично, $inv_\tau.p$ определяет, что p выполнено для любого исходного состояния и сохраняется при каждом атомном переходе. Таким образом, по индукции, p является выполненным (верным) для любого состояния. Важно заметить, что $[\mathcal{W}.F.q \Longrightarrow q]$ поскольку $next_\tau.(p,q).F \Longrightarrow [p \Longrightarrow q]$.

Согласно приведенному выше определению, индуктивный инвариант означает, что существует строгое соответствие между элементами двух множеств объектов, которые связаны посредством некоторого отношения (преобразования). Такое определение очень близко к реляционному подходу к определению преобразования модели, когда объявляются отношения между объектами (и ссылками) исходного и целевого языков. Это приводит к идее, что индуктивный инвариант может быть эффективным механизмом для определения таких преобразований модели и для проверки возможности получения одной модели путем преобразования другой.

Операционный же инвариант может быть определен в терминах вычислений над транзитивной моделью [44]. Система переходов F может быть связана с

подмножеством $\mathcal{O}.F$ множества $(\Sigma, \mathrm{F})^{\omega}$ бесконечных последовательностей состояний следующим образом: бесконечное вычисление $\sigma = \langle \sigma_0, \sigma_1, \dots, \sigma_n, \dots \rangle$ принадлежит множеству $\mathcal{O}.F$ тогда и только тогда, когда:

1. $\mathcal{J}.F.\sigma_0$
2. $\forall i \in \mathbb{N}: \mathcal{W}.F.\{\sigma_{i+1}\}.\sigma_i$, где $\{\sigma_{i+1}\}$ предикат состояния, который принимает значение true для состояния σ_{i+1} и false для любого другого состояния.

С неформальной точки зрения, $\mathcal{O}$ состоит из тех последовательностей состояний, которые начинаются с начального состояния, удовлетворяющего $\mathcal{J}$ и в котором каждое последующее состояние является результатом перехода $\mathcal{W}$. Множество $\mathcal{O}$ является непустым, поскольку $\mathcal{J}$ выполнимо и $\mathcal{W}$ включает последовательные, связанные шаги.

После того, как вычисления переходной системы построены, спецификации *next* и *inv* определяются следующим образом:

$$next_{\mathcal{O}}.(p,q).F \triangleq \forall \sigma \in \mathcal{O}.F: \forall i \in \mathbb{N}: p, \sigma_i \Longrightarrow q, \sigma_{i+1} \quad (4)$$

$$inv_{\mathcal{O}}.p.F \triangleq \forall \sigma \in \mathcal{O}.F: \forall i \in \mathbb{N}: p, \sigma_i \quad (5)$$

В данном случае, $next_{\mathcal{O}}.(p,q)$ означает, что при любом вычислении системы за любым состоянием, удовлетворяющим p всегда следует состояние, удовлетворяющее q. Хотя вычисления включают связанные (последовательные) преобразования, $next_{\mathcal{O}}.(p,q)$ не означает, что $[p \Rightarrow q]$. По аналогии, $inv_{\mathcal{O}}.p$ означает, что любое состояние любого вычисления системы удовлетворяет p. Очевидно, что $next_{\mathcal{O}}$ и $inv_{\mathcal{O}}$ связаны таким же образом, как отношения между $next_{\tau}$ and inv_{τ}, а именно: $inv_{\mathcal{O}}.p.F \equiv next_{\mathcal{O}}.(p,q).F \wedge [\mathcal{J}.F \Longrightarrow p]$.

Важно заметить, что в нашем случае используются все виды инвариантов, как объектные, так и функциональные. В соответствии с этим, процесс построения интерфейса на основе метамодели ПОЯ выглядит следующим образом.

1. Выделяются объектные инварианты на уровне метамодели ПОЯ (в данном случае это – отдельные атрибуты соответствующих объектов, которые в дальнейшем должны быть отражены на интерфейсе).

2. Выделяются инварианты на уровне интерфейса (это – мини-шаблоны, которые отвечают за отображение каждого отдельного атрибута метамодели в зависимости от его названия и типа).
3. Определяется функция проекции $f: A \to B$, где A и B являются инвариантами, выделенными в п. 1 и 3 соответственно на уровнях метамодели и интерфейса.

В результате процесс построения интерфейса в виде композиции инвариантов можно описать в псевдокоде следующим образом:

```
...
foreach metamodel_invariant do
  foreach interface_invariant do
    if mapping_function(metamodel_invariant) == interface_invariant then
      add interface_invariant onto GUI
...
```

Такой подход является эффективным, поскольку позволяет автоматизировать процесс построения интерфейсов, а также обеспечить их согласованную эволюцию в случае внесения изменений в базовую метамодель ПОЯ. Автоматизация обеспечивается тем, что, установив соответствия на уровне объектных инвариантов, мы определили функцию проекции в виде индуктивного инварианта. Как следствие, структура преобразований является устойчивой к изменениям, поскольку меняются не инварианты, а только множество объектов метамодели, являющихся композицией объектных инвариантов.

Как следствие, внесение изменений в интерфейс больше не требует его ручной перерисовки, а проводится в реальном режиме времени [43].

Оценка эффективности данного подхода в работе оценивается на примере разработки программной среды отчетности для сотрудника приемной комиссии ВУЗа. В процессе разработки программной среды было написано 10608 строк кода (на языке Java), 10 графических мини-шаблонов для различных доменов данных реляционной SQL базы данных, включающей 47 связанных таблиц (число отношений связи - 79) с общим количеством столбцов 455. По результатам оценки внедрения, время адаптации отчетных форм к изменениям в правилах приема

сократилось, поскольку ранее требовало минимум 24 часов (время, регламентированное соглашением с разработчиками), а при применении проекционного подхода ограничено лишь временем, необходимым для изменения метамодели ПОЯ, что в среднем составляет от 15 минут до 1 часа.

Такая эффективность связана не только с применением проекционного подхода, но также и со структурой самого программного продукта, все компоненты которого строятся на принципе композиции из меньших фрагментов, отвечающих за отдельный функционал (и составляющих инвариант). Как следствие, при изменении структуры базы данных изменения в интерфейс происходят автоматически, поскольку его структура не является статической, а является результатом применения функции соответствия, которая по инварианту на уровне данных отображает в интерфейсе соответствующий графический инвариант. Как следствие, программная среда является адаптивной и очень гибкой и не требует перерисовки и пересоздания интерфейсе при изменении структуры данных, к которой интерфейс привязан. Все изменения происходят автоматически, за счет чего и достигается значительное сокращение времени изменения программной среды в целом.

АПРОБАЦИЯ РЕЗУЛЬТАТОВ ИССЛЕДОВАНИЯ

Основные результаты диссертации докладывались и обсуждались на следующих конференциях и семинарах:

4. The 18th International Conference on Perspectives in Business Informatics Research (BIR 2019), September 23-25, 2019, Katowice, Poland.
5. 9th IFAC Conference on Manufacturing Modelling, Management and Control, August 28-30, 2019, Berlin, Germany.
6. Master & Doctoral Consortium at 15th International Workshop on Enterprise & Organizational Modeling and Simulation (EOMAS), June 3-4, 2019, Rome, Italy.
7. The 17th International Conference on Perspectives in Business Informatics Research (BIR 2018), Septermber 24-26, 2018, Stockholm, Sweden.

8. The 22nd Conference of Open Innovations Association FRUCT May 15-18, 2018, Jyvaskyla, Finland.
9. Городской семинар «ИНФОРМАТИКА И АВТОМАТИЗАЦИЯ» Санкт-Петербургского института информатики и автоматизации Российской Академии наук (СПИИРАН) (8 декабря 2017г.).
10. The 16th International Conference on Perspectives in Business Informatics Research (BIR 2017), August 28–30, 2017, Copenhagen, Denmark.
11. The 15th International Conference on Perspectives in Business Informatics Research (BIR 2016), September 15–16, 2016, Prague, Czech Republic.

Список публикаций по тематике исследования включает в себя 10 работ, перечисленных ниже. Из них 8 опубликованы в изданиях, индексируемых Scopus (2-7, 9-10), а также одна – в журнале из списка ВАК (1).

1. Моделирование динамики онлайн-дискуссий в сети Интернет с использованием многоагентных систем / Бабкин Э.А., Бабкина Т.С., Улитин Б.И. // Бизнес-информатика. – № 2 (44). – 2018. – с. 17–29.
2. Providing Models of DSL Evolution Using Model-to-Model Transformations and Invariants Mechanisms / B. Ulitin, E. Babkin // Digital Transformation and New Challenges, Lecture Notes in Information Systems and Organisation 40. Switzerland: Springer, 2020. P. 37-48.
3. Ontology-based reconfigurable DSL for planning technical services / B. Ulitin, E. Babkin // IFAC-PapersOnLine. 2019. Vol. 52. No. 13. P. 1138-1144.
4. Digitalization: A meeting point of knowledge management and enterprise engineering / E. Babkin, T. Poletaeva, B. Ulitin //IC3K 2019 - Proceedings of the 11th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management. Vol. 3. SciTePress, 2019. P. 22-36.
5. Automated formal verification of model transformations using the invariants mechanism / Boris Ulitin, Eduard Babkin, Tatyana Babkina, Arsenii Vizgunov // Perspectives in Business Informatics Research. 18th International Conference, BIR 2019, Katowice, Poland, September 23-25, 2019 Proceedings. Lecture Notes in

Business Information Processing. Issue 365. Switzerland: Springer, 2019. P. 59-73.

6. A Projection-Based Approach for Development of Domain-Specific Languages / B. Ulitin, E. Babkin, T. Babkina // Perspectives in Business Informatics Research. 17th International Conference, BIR 2018, Stockholm, Sweden, September 24-26, 2018 Proceedings. Lecture Notes in Business Information Processing. Issue 330. Switzerland: Springer, 2018. P. 219-234.
7. An Object-Oriented Model for Smart Devices in Internet of Things / B. Ulitin, E. Babkin // Proceedings of the 22st Conference of Open Innovations Association FRUCT, Jyvaskyla, Finland. – 2018. – ISBN 978-952-68653-4-8. – p. 263-271.
8. Ontology-based DSL development using graph transformations methods / B. Ulitin, E. Babkin, T. Babkina // Journal of Systems Integration. – v.9 (2). – pp.37-51. – 2018. – ISSN: 1804-2724.
9. Ontology and DSL co-evolution using graph transformations methods / B. Ulitin, E. Babkin // Perspectives in Business Informatics Research. 16th International Conference, BIR 2017, Copenhagen, Denmark, August 28-30, 2017 Proceedings. Lecture Notes in Business Information Processing. Issue 295. Switzerland: Springer, 2017. P. 233-247.
10. Combination of DSL and DCSP for decision support in dynamic contexts / B. Ulitin, E. Babkin, T. Babkina // Perspectives in Business Informatics Research. 15th International Conference, BIR 2016, Prague, Czech Republic, September 15-16, 2016 Proceedings. Lecture Notes in Business Information Processing. Issue 261. Switzerland: Springer, 2016. P. 159-173.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ

1. Ануреев, И.С. Предметно-ориентированные системы переходов: объектная модель и язык // Системная информатика. – 2013. – №. 1. – с. 1-34.
2. Гайфуллин, Б.Н., Туманов, В.Е. Предметно-ориентированные системы научной осведомленности в науке и образовании // Современные информационные технологии и ИТ-образование. – 2012. – №. 8. – с. 741-750.
3. Конструирование канонических информационных моделей для интегрированных информационных систем / В. Н. Захаров, Л. А. Калиниченко, И. А. Соколов, С. А. Ступников // Информатика и ее применения. – 2007. – Т. 1, № 2. – с. 15-38.
4. Сухов, А.О. Классификация предметно-ориентированных языков и языковых инструментариев // Математика программных систем. – 2012. – с. 74-83.
5. Федоренков, В.Г., Балакшин, П.В. Особенности применения предметно-ориентированных языков для тестирования веб-приложений // Программные продукты и системы. – 2019. – №4. – с. 601-606.
6. Agrawal, A., Karsai, G., Shi, F.: Graph Transformations on Domain-Specific Models. In: International Journal on Software and Systems Modeling, pp. 1-43. Nashville: Vanderbilt University Press (2003).
7. Akehurst, D., Kent, S.: A relational approach to defining transformations in a metamodel. In J.-M. Jézéquel, H. Hussmann, and S. Cook (eds.), Proc. Fifth International Conference on the Unified Modeling Language – The Language and its Applications, LNCS, vol. 2460, pp. 243–258. Springer-Verlag, Dresden, Germany (2002).
8. Bell, P. Automated Transformation of Statements within Evolving Domain Specific Languages // Computer Science and Information System Reports. – 2007. – pp. 172–177.
9. Bergmann, G., Ráth, I., Varró, G., Varró, D.: Change-driven model transformations. Software & Systems Modeling 11(3), pp. 431-461 (2012).

10. Cabot, J., Clarisó, R., Guerra, E., de Lara J. Verification and validation of declarative model-to-model transformations through invariants. – J Syst Softw 83(2), 283–302 (2010).
11. Challenger, M., Demirkol, S., Getir, S., Mernik, M., Kardas, G., Kosar, T.: On the use of a domain-specific modeling language in the development of multiagent systems. In: Engineer-ing Applications of Artificial Intelligence, pp. 111-141 (2014).
12. Chen, Y., Tang, Z.: Vector invariant fields of finite classical groups. In: Journal of Algebra, v. 534, pp. 129-144 (2019).
13. Cleenewerck, T. Component-Based DSL Development // Generative Programming and Component Engineering. Second International Conference, GPCE 2003, Erfurt, Germany, September 22-25, 2003. Proceedings. – 2003. – pp. 245-264.
14. Cleenewerck, T. Evolution and Reuse of Language Specifications for DSLs (ERLS) / T. Cleenewerck, K. Czarnecki, J. Striegnitz, M. Volter // Object-Oriented Technology. ECOOP 2004 Workshop Reader. – 2004. – pp. 187-201.
15. Demuth, A., Riedl-Ehrenleitner, M., Lopez-Herrejon, R.E., and Egyed, A. (2016). Co-evolution of metamodels and models through consistent change propagation. Journal of Systems and Software, pp. 281–297.
16. Eclipse Graphical Modeling Project (GMP) http://www.eclipse.org/modeling/gmp/.
17. Evans, E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley (2013)
18. Fernandez-Lopez, M., Gomez-Perez, A.: Overview and analysis of methodologies for building ontologies. In: The Knowledge Engineering Review, pp. 129-156. Cambridge University Press (2002)
19. Fowler, M. Domain specific languages / M. Fowler, R. Parsons. – Addison Wesley, 2010. – 413 p. – ISBN: 978-0-321-71294-3.
20. Gómez-Abajo, P. A domain-specific language for model mutation and its application to the automated generation of exercises / Pablo Gómez-Abajo, Esther Guerra, Juan de Lara // Computer Languages, Systems & Structures. – 2016.

21. Guizzardi, G.: Ontological foundations for structural conceptual models. Telematica Instituut Fundamental Research Series, No. 15, ISBN 90-75176-81-3, The Netherlands (2005).
22. Guizzardi, G., Halpin, T.: Ontological Foundations for Conceptual Modeling. In: Applied Ontology, v.3, p. 91-110 (2008).
23. Haav, H.-M., Ojamaa, A., Grigorenko, P., Kotkas, V.: Ontology-Based Integration of Software Artefacts for DSL Development. In: On the Move to Meaningful Internet Systems: OTM 2015 Workshops. Lecture Notes in Computer Science, v. 9416, pp.309-318. Springer (2015).
24. Kelly, S. Domain-specific modeling: enabling full code generation [Text] /S. Kelly, J.-P. Tolvanen. — Hoboken, New Jersey, USA: Wiley-IEEE ComputerSociety Press, 2008. — P. 444. — ISBN:978-0-470-03666-2.
25. Kessentini, W., Sahraoui, H., and Wimmer, M. (2019). Automated metamodel/model co-evolution: A search-based approach. Information and Software Technology, pp. 49–67.
26. Kogalovsky M. R., Kalinichenko L. A. Conceptual and ontological modeling in information systems // Программирование. — 2009. — Vol. 35, no. 5. — P. 241–256.
27. Kosar, T. Domain specific languages: A systematic mapping study / T. Kosar, S. Bohra, M. Mernik // Information and Software Technology. – 2016. – v. 71. – pp. 77–91.
28. Kosar, T., Martınez Lopez, P., Barrientos, P., Mernik, M.: A preliminary study on various implementation approaches of domain-specific language. In: Information and Software Technology, pp.390-405. Elsevier (2008).
29. Laird, P. Towards Dynamic Evolution of Domain Specific Languages / P. Laird, S. Barrett // Software Language Engineering. – 2010. – pp. 144-153.
30. Mengerink, J.G.M. Udapt Edapt Extensions for Industrial Application / Josh G.M. Mengerink, Alexander Serebrenik, Mark van den Brand, Ramon R.H. Schiffelers // ITSLE 2016 Industry Track for Software Language Engineering October 31, 2016, Amsterdam, Netherlands. – 2016. – pp. 21-22.

31. Mengerink, J.G.M. A Complete Operator Library for DSL Evolution Specification / J.G.M. Mengerink, A. Serebrenik, R.R.H. Schiffelers, M.G.J. van den Brand // MDSE 32nd International Conference on Software Maintenance and Evolution, 2016. – 2016. – pp. 144-154.
32. Mernik, M. When and how to develop domain specific languages / M. Mernik, J. Heering, AM Sloane // ACM computing surveys (CSUR). – 2005. – v. 37(4). – pp. 316–344.
33. MetaLanguage http://pespmc1.vub.ac.be/METALARE.html.
34. Mohagheghi, P. Evaluating Domain-Specific Modelling Solutions / Parastoo Mohagheghi and Øystein Haugen // ER 2010 Workshops. – 2010. – pp. 212-221.
35. Parr, T. (2012). Language Implementation Patterns: Create Your Own Domain-Specific and General Programming Languages. Pragmatic Bookshelf.
36. Pereira, M.J.V. Ontological approach for DSL development / Maria João Varanda Pereira, João Fonseca, Pedro Rangel Henriques // Computer Languages, Systems & Structures. – 2016. – v. 45. – pp. 35-52.
37. Popovic, A. A DSL for modeling application-specific functionalities of business applications / Aleksandar Popovic, Ivan Lukovic, Vladimir Dimitrieski, Verislav Djuki // Computer Languages, Systems & Structures. – 2015. – pp. 69-95.
38. Ruffolo, M., Sidhu, I., Guadagno, L.: Semantic Enterprise Technologies. In: Proceedings of the First International Conference on Industrial Results of Semantic Technologies, v. 293, pp.70-84 (2007).
39. Sprinkle, J. A domain-specific visual language for domain model evolution / Jonathan Sprinkle, Gabor Karsai // Journal of Visual Languages & Computing. – 2004. – v. 15, issue 3-4. – pp. 291–307.
40. Ulitin, B., Babkin, E.: Ontology and DSL co-evolution using graph transformations methods. In: Lecture Notes in Business Information Processing Issue 295: Perspectives in Business Informatics Research, pp. 233-247. Springer, Switzerland (2017).
41. Ulitin, B., Babkin, E., Babkina, T.: A Projection-Based Approach for Development of Domain-Specific Languages. In: Lecture Notes in Business Information

Processing Issue 330: Perspectives in Business Informatics Research, pp. 219-234. Springer, Switzerland (2018).

42. Ulitin, B., Babkin, E., Babkina, T., Vizgunov, A.: Automated Formal Verification of Model Transformations Using the Invariants Mechanism. In: Lecture Notes in Business Information Processing Issue 365: Perspectives in Business Informatics Research, pp. 59-73. Springer, Switzerland (2019).
43. Ulitin, B., Babkin, E.: Ontology-based reconfigurable DSL for planning technical services // IFAC-PapersOnLine. 2019. Vol. 52. No. 13. P. 1138-1144.
44. Ulitin, B., Babkin, E.: Providing Models of DSL Evolution Using Model-to-Model Transformations and Invariants Mechanisms // Digital Transformation and New Challenges, Lecture Notes in Information Systems and Organisation 40. Switzerland: Springer, 2020. P. 37-48.