На правах рукописи

Бреслав Андрей Андреевич

МЕХАНИЗМЫ КОМПОЗИЦИИ В ПРЕДМЕТНО-ОРИЕНТИРОВАННЫХ ЯЗЫКАХ

Специальность 05.13.11 — Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

АВТОРЕФЕРАТ
диссертации на соискание ученой степени
кандидата технических наук

Санкт-Петербург — 2010

Работа выполнена в Санкт-Петербургском государственном университете информационных технологий, механики и оптики (СПбГУ ИТМО).

Научный руководитель:	доктор физико-математических наук, профессор Попов Игорь Юрьевич
Официальные оппоненты:	?
	?
Ведущая организация:	Санкт-Петербургский Академический университет – научно-образовательный центр нанотехнологий РАН

Защита диссертации состоится “ ” 2010 г. в час. на заседании диссертационного совета Д 212.227.06 при Санкт-Петербургском государственном университете информационных технологий, механики и оптики по адресу: 197101, Санкт-Петербург, Кронверкский пр., д. 49.

С диссертацией можно ознакомиться в библиотеке Санкт-Петербургского государственного университета информационных технологий, механики и оптики.

Автореферат разослан “ ” 2010 г.

Ученый секретарь диссертационного совета, доктор технических наук, профессор Лисицына Л. С.

ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ

Актуальность темы. Развитие технологии программирования исторически идет по пути повышения уровня абстракции поддерживаемого инструментами, используемыми при разработке программного обеспечения (ПО). Решающим шагом на этом пути явилось создание языков программирования высокого уровня, позволяющих лишь в небольшой степени заботиться об особенностях конкретной аппаратной архитектуры. Повышение уровня абстракции — один из ключевых факторов, определяющих сокращение сроков создания программных средств, поскольку высокоуровневые инструменты позволяют избегать определенных типов ошибок и повторно использовать разработанные решения, а также облегчают командную разработку.

С развитием языков высокого уровня неразрывно связан процесс развития автоматизированных инструментов разработки трансляторов, базирующийся на достижениях теории формальных языков и грамматик, в частности, алгоритмах преобразования контекстно-свободных грамматик в магазинные автоматы и формализация семантики с помощью атрибутных грамматик.

Дальнейшее повышение уровня абстракции привело к возникновению идеи *предметно-ориентированных языков* (ПОЯ), предназначенных для решения задач в относительно узкой предметной области и нередко непригодных за ее пределами. ПОЯ противопоставляются языкам общего назначения, являющимся вычислительно универсальными и позволяющим решать любые задачи. Основной мотивацией к разработке и использованию ПОЯ является тот факт, что моделирование предметной области в языках общего назначения часто бывает недостаточно явным, что приводит к большим объемам кода и затруднениям при чтении программ. При использовании ПОЯ эта проблема снимается, поскольку такие языки оперируют непосредственно понятиями предметной области и могут даже позволить специалистам в этой области, не имеющим квалификации разработчиков ПО, принимать участие в написании программ. В настоящее время ПОЯ применяются во множестве областей, начиная с систем управления базами данных и заканчивая системами моделирования бизнес-процессов.

Использование ПОЯ снижает затраты на разработку ПО в данной предметной области, но разработка самих ПОЯ также требует затрат. Если эти затраты высоки, то использование ПОЯ может быть нецелесообразным, поэтому возникает задача автоматизации разработки таких языков с целью минимизировать затраты на их создание и поддержку. Традиционные

средства разработки трансляторов не обеспечивают необходимый уровень автоматизации, поэтому разрабатываются новые подходы и инструменты, позволяющие быстро разрабатывать небольшие языки с поддержкой все более сложных механизмов. В частности, существенный интерес представляют механизмы композиции, обеспечивающие повторное использование кода, написанного на ПОЯ.

Предметом исследования являются механизмы композиции, пригодные для использования в предметно-ориентированных языках.

Целью работы является исследование и обоснование подходов и методов, позволяющих автоматически расширять предметно-ориентированные языки механизмами композиции, поддерживающими повторное использование.

Задачи исследования. Достижение поставленной цели подразумевает решение следующих задач:

- Проектирование и реализация предметно-ориентированного языка для хорошо изученной области — описания текстового синтаксиса искусственных языков — поддерживающего все основные механизмы композиции в полном объеме, с целью выявления связей между этими механизмами и их характерных особенностей, влияющих на подходы к автоматизации.
- Проверка адекватности разработанного языка нуждам конечных пользователей на примере описания синтаксиса сложных языков.
- Обобщение рассмотренных механизмов композиции в виде формализованных языковых конструкций. Описание их семантики и соответствующих систем типов.
- Разработка алгоритмов автоматического расширения языка механизмами композиции, обоснование их корректности.
- Применение предложенного подхода к существующему предметно-ориентированному языку.

Методы исследования включают методы инженерии программного обеспечения, анализа алгоритмов и программ, аппарат теории типов, теории графов и теории формальных грамматик.

Научная новизна результатов работы состоит в том, что:

- Спроектирован и реализован предметно-ориентированный язык для описания текстового синтаксиса, поддерживающий композицию спе-

цификаций с помощью модулей, шаблонов (типизированных макроопредений) и аспектов.

- На основе указанного языка разработан генератор трансляторов, поддерживающий проверку типов в семантических действиях и гарантирующий отсутствие ошибок типизации в сгенерированном коде для многих языков реализации.

- Предложена формализация механизмов композиции на основе шаблонов и аспектов, и доказаны свойства данной формализации, гарантирующие раннее обнаружение ошибок программиста при использовании этих механизмов.

- Разработаны и апробированы алгоритмы, позволяющие автоматизировать расширение предметно-ориентированных языков механизмами композиции, основанными на шаблонах и аспектах.

Практическую ценность работы составляют:

- Разработанная библиотека, обеспечивающая трансляцию предложенного языка описания текстового синтаксиса.

- Программные генераторы, использующие данную библиотеку, позволяющие автоматически получать трансляторы и компоненты интегрированной среды разработки.

- Методика и алгоритмы расширения предметно-ориентированных языков механизмами композиции.

На защиту выносятся следующие положения:

- Предметно-ориентированный язык для описания текстового синтаксиса, поддерживающий композицию спецификаций с помощью модулей, шаблонов и аспектов.

- Подход к описанию и генерации трансляторов, позволяющий порождать код на нескольких языках программирования по одной спецификации, и гарантирующий отсутствие ошибок типизации в сгенерированном коде.

- Метод автоматического расширения имеющихся описаний синтаксиса и семантики предметно-ориентированного языка таким образом, что результирующий язык поддерживает композицию с помощью шаблонов и аспектов.

Достоверность научных результатов и выводов обеспечивается формальной строгостью описания процесса композиции языков, обоснованностью применения математического аппарата, результатами тестирования алгоритмов и программного обеспечения.

Внедрение результатов работы. Результаты, полученные в ходе диссертационной работы, были использованы в компании OpenWay (Санкт-Петербург) при разработке предметно-ориентированного языка для написания отчетов, а также при выполнении НИОКР "Технология разработки предметно-ориентированных языков" по программе "У.М.Н.И.К." Фонда содействия развитию малых форм предприятий в научно-технической сфере.

Апробация работы. Изложенные в диссертации результаты обсуждались на 11 российских и международных научных конференциях, семинарах и школах, включая V, VI и VII всероссийские межвузовские научные конференции молодых ученых (2008, 2009 и 2010 гг., Санкт-Петербург), международную научную конференцию "Компьютерные науки и информационный технологии" (2009 г., Саратов), международные научные школы "Generative and Transformational Techniques in Software Engineering" (2009 г., Брага, Португалия), "Aspect-Oriented Software Development" (2009 г., Нант, Франция) и "15^{th} Estonian Winter School in Computer Science" (2010 г., Палмсе, Эстония), а также международные семинары "Teooriapäevad" (2009 и 2010 гг., Эстония), семинар Лаборатории математической логики и семантики языков программирования Научно-исследовательского института кибернетики Эстонской Академии наук (2009 г., Таллинн, Эстония) и научном семинаре "Computer Science Клуба" при ПОМИ РАН (2009 г., Санкт-Петербург).

Публикации. По теме диссертации опубликовано пять печатных работ (из них две статьи — в изданиях, соответствующих требованиям ВАК РФ к кандидатским диссертациям по данной специальности).

Структура и объем работы. Диссертация состоит из введения, четырех глав, списка литературы (? наименований) и 3 приложений. Содержит ? с. текста (из них ? основного текста и ? — приложений), включая ? рис. и табл.

СОДЕРЖАНИЕ РАБОТЫ

Во введении обосновывается актуальность проблемы, формулируются цель и задачи исследования, отмечаются научная новизна и практическая значимость результатов, перечисляются основные положения, выносимые на защиту.

В первой главе сообщаются предварительные сведения об инженерии языков и приводится обзор литературы. В частности, описываются основные понятия концепции *мета-моделирования*; изложение базируется на мета-мета-модели Ecore, входящей в библиотеку EMF. Для удобства дальнейшего использования вводится текстовая нотация для моделей и приводятся правила систем типов, отражающих требования мета-моделей.

Далее приводится обзор разновидностей конкретного синтаксиса предметно-ориентированных языков. Рассматривается текстовый синтаксис на основе контекстно-свободных грамматик и на основе XML, а также основные виды графического синтаксиса: диаграммы, древовидное представление моделей и псевдо-текстовый синтаксис. Выделяются сходства и различия данных видов синтаксиса; в последующих частях работы рассматривается в основном текстовый синтаксис, что обусловлено его универсальностью и сравнительно низкими требованиями к инструментальной поддержке.

Приводится обзор средств разработки текстового синтаксиса, причем основное внимание уделяется механизмам композиции, использованным в соответствующих инструментах: модулям, параметризованным конструкциям (шаблонам и макроопределениям) и аспектам. Проводится сравнительный анализ реализаций этих механизмов, позволяющий сделать вывод о том, что поддержка более сложных из них в большинстве инструментов ограничена и нуждается в расширении.

Также приводится обзор средств автоматизации разработки самих механизмов композиции. Их сравнительный анализ показывает, что подходы, позволяющие автоматически расширять текстовые языки поддержкой механизмов композиции, нуждаются в дальнейшем улучшении.

Во второй главе описывается предметно-ориентированный язык Grammatic, предназначенный для описания текстового синтаксиса и поддерживающий композицию с помощью модулей, шаблонов и аспектов. Этот язык предназначен для использования в качестве обобщенной расширяемой нотации для контекстно-свободных грамматик, которую можно применять при разработке различных инструментов. Реализация данного языка представляет собой библиотеку, позволяющую транслировать текстовые

описания грамматик во внутреннее объектно-ориентированное представление на основе библиотеки EMF, к которому имеется открытый программный интерфейс. Различные инструменты, такие как генераторы трансляторов и других программных средств, анализаторы грамматик и т.д., могут использовать данный язык как единый формат для ввода данных, что позволит разработчикам сосредоточиться на реализации функциональности, специфичной для их задачи.

Язык Grammatic основан на нотации EBNF, использующей расширенный набор операций для описания контекстно-свободных правил, что позволяет в той же нотации описывать и лексические анализаторы. Поскольку многим инструментам, кроме правил грамматики, требуется дополнительная информация, такая как код семантических действий или приоритеты бинарных операций, Grammatic позволяет оснащать любые элементы грамматики *метаданными* в виде аннотаций, которые не интерпретируются самой библиотекой и просто преобразуются во внутреннее представление для дальнейшего использования. Аннотации состоят из пар "имя-знаячение", где значения могут иметь различные типы, в том числе и определяемые пользователем. Механизм метаданных позволяет использовать предложенный язык для решения широкого круга задач.

Основным достоинством Grammatic является наличие поддержки шаблонов и аспектов. *Шаблоны* (типизированные макроопределения) представляют собой фрагменты грамматик, содержащие формальные параметры. Результатом применения шаблона является подстановка конкретных элементов грамматики вместо формальных параметров. Такой механизм позволяет единообразно описывать конструкции, часто используемые в грамматиках разных языков. Например, списки с разделителями описываются следующим шаблоном:

```
template List<element : Expression, sep : Expression> : Expression {
    <?element> (<?sep> <?element>)*
}
```

В угловых скобках после имени `List` указаны формальные параметры с типами (типы обычно можно не указывать, здесь они приведены для наглядности), в фигурных скобках записано тело шаблона. Шаблон можно использовать, указав в угловых скобках его имя и аргументы, занимающие места формальных параметров, например, выражение `<List INT, '*'>` будет развернуто в `INT ('*' INT)*`. Результатом применения шаблона может быть не только грамматическое выражение, но и любая другая конструкция языка Grammatic, например, аннотация или одно или несколь-

ко правил грамматики. Это позволяет реализовать с помощью шаблонов концепцию *параметрических модулей*.

Аспекты в языке GRAMMATIC реализуются следующим образом:

- *Точками встраивания* (join points) являются все элементы языка GRAMMATIC, такие как символы, различные выражения, продукции и аннотации.

- *Срезы* (point-cuts), обеспечивающие квантификацию (quantification), описываются в виде образцов (patterns), использующих переменные, с которыми связываются конкретные фрагменты сопоставляемых выражений. Кроме того, используются *подстановочные знаки*, соответствующие произвольным конструкциям одного определенного типа (например, произвольным выражениям или произвольным символам).

- *Советы* (advice), описывающие изменения, привносимые в точках встраивания, позволяют встраивать результаты разворачивания данных шаблонных выражений до, после или вместо фрагментов точек встраивания, а также добавлять аннотации к элементам грамматики.

Приведем пример *аспектного правила*, использующего данные понятия:

```
example : .. e=example  .. ;    // Срез
    instead ?e : '(' <?e> ')' ; // Совет
```

Данный пример заменяет рекурсивное вхождение символа `example` в правой части на тот же символ, заключенный в скобки.

С помощью аспектов достигается повторное использование и расширение спецификаций, в которых не предусмотрены такие возможности (например, эти спецификации не являются шаблонными). Кроме того, этот механизм позволяет разделять спецификацию на фрагменты с различным назначением, например, отделять метаданные для различных генераторов (в частности, семантические действия) от грамматических правил, значительно улучшая читаемость и модульность спецификаций.

В Таб. 1 приводятся результаты сравнения механизмов композиции GRAMMATIC с соответствующими механизмами в других инструментах. Из таблицы видно, что GRAMMATIC поддерживает шаблоны и аспекты значительно лучше других инструментов. Это делает целесообразным использование данного языка для описания грамматик. Существующие проекты, в свою очередь, реализуют сложную функциональность, связанную с автоматизацией построения трансляторов и других инструментов. GRAMMATIC

	Yacc/Bison	ANTLR	Menhir	xText	ASF+SDF	JastAdd	AspectG	Silver	LISA	Rats!	**Grammatic**
Шаблоны грамматик. Параметры:											
символы	-	-	-	-	+	-	-	-	-	-	+
модули	-	-	-	-	-	-	-	-	-	+	-
произвольные выражения	-	-	-	-	-	-	-	-	-	-	+
Шаблоны выражений. Параметры:											
символы	-	-	+	-	-	-	-	-	-	-	+
произвольные выражения	-	-	-	-	-	-	-	-	-	-	+
Шаблоны сем. действий	-	-	-	-	-	-	-	-	+	-	+
Аспекты											
Незнание	-	-	-	-	-	+	+	+	+	-	+
Квантификация	-	-	-	-	-	-	+	+	+	-	+
Гибкость при повторном использовании											
Добавление продукций	-	+	+	+	+	+	+	+	+	+	+
Замена элементов	-	+	-	+	-	-	-	-	-	+	+
Удаление элементов	-	-	-	-	-	-	-	-	-	+	+
Изменение метаданных	-	-	-	+	+	+	+	+	+	+	+

Таблица 1: Сравнение Grammatic с существующими инструментами

достаточно легко интегрируется с подобными инструментами, что позволяет пользоваться как преимуществами шаблонов и аспектов, так и уже наработанными достижениями в других областях.

Предложенный язык был использован для декларативного описания компонент интегрированных сред разработки (подсветки синтаксиса и автоматического форматирования кода), для описания синтаксиса самого Grammatic, а также семейства диалектов языка SQL92. В последнем случае механизм аспектов позволил сделать описание очень кратким, поскольку для описания диалектов достаточно обозначить его отличия от стандартного варианта языка.

Третья глава посвящена описанию генератора трансляторов Grammatic^{SDT}, построенного на базе языка Grammatic. Отличительной особенностью данного инструмента является механизм проверки типов в семантических действиях, обеспечивающий раннее обнаружение ошибок. Практически все существующие генераторы не выполняют таких проверок, что приводит к появлению ошибок компиляции в сгенерированном коде и

затрудняет разработку. Кроме того, имеется возможность генерации кода на нескольких *языках реализации*[1] по одной спецификации, что важно для разработки предметно-ориентированных языков, поскольку эти языки не должны ограничивать разработчиков в выборе платформы или языка общего назначения. Результаты сравнения возможностей GRAMMATIC^{SDT} с другими инструментами приведены в Таб. 2.

	COCO/R	YACC/BISON	ELI	ANTLR	GOLD	SABLECC	GRAMMATIC^{SDT}
Генерация кода на разных языках	+	+	-	+	+	+	+
по одной спецификации	-	-	-	-	+	+	+
Семантические действия в спецификациях	+	+	+	+	-	-	+
Ошибки в терминах спецификации	-	-	+	-	n/a	n/a	+
Проверка типов в спецификациях	-	-	-	-	n/a	n/a	+

Таблица 2: Сравнение GRAMMATIC^{SDT} с существующими инструментами

GRAMMATIC^{SDT} принимает на вход грамматику, аннотированную семантическими действиями, задающими схему L-атрибутной трансляции, выполняет статические проверки и генерирует транслятор, используя один из существующих инструментов (например, ANTLR). Для различных языков реализации могут быть использованы различные инструменты. Семантические действия пишутся на абстрактном языке, поддерживающем следующие операции: (а) чтение и запись атрибута, (б) вызов внешней функции, написанной на языке реализации, (в) передача параметров функциям, осуществляющим разбор. По спецификации на языке GRAMMATIC^{SDT} генерируется транслятор и заголовки использованных внешних функций на языке реализации.

Стандартные правила системы типов для присваиваний, описания и вызовов функций в случае GRAMMATIC^{SDT} параметризованы отношением “подтип-супертип”, обозначаемым и регламентирующим полиморфизм, и множеством конкретных *базовых типов*, среди которых специально выделяется тип символьных строк . Механизмы проверки и реконструкции типов работают для произвольных значений этих параметров и имеют открытый программный интерфейс, позволяющий создавать расширения, поддерживающие различные языки реализации. В частности, автором раз-

[1] Язык программирования, код на котором является результатом работы генератора.

работано расширение, интегрирующее GRAMMATIC^{SDT} с системой типов языка JAVA.

Кроме возможности создания расширений системы типов для интеграции с конкретными языками реализации, разработано и "универсальное" расширение, позволяющее декларативно описывать множество базовых типов и отношение . Такого описания (называемого системой типов *абстрактного уровня*) достаточно для проверки и вывода типов в спецификациях. Для правильно работы генератора необходимо указать, как типы абстрактного уровня отображаются на типы языка реализации. Декларативное описание этого отображения называется системой типов *конкретного уровня*. Таким образом, для данной спецификации существует ровно одна система типов абстрактного уровня и, возможно, несколько систем конкретного уровня — как минимум, по одной на каждый язык реализации. Описания конкретного уровня можно добавлять, не изменяя спецификации.

Транслятор языка GRAMMATIC был реализован с помощью GRAMMATIC^{SDT}: основным языком реализации является JAVA, а для генерации синтаксического анализатора используется ANTLR.

В четвертой главе описан метод, позволяющий автоматически расширять предметно-ориентированные языки механизмами композиции, основанными на шаблонах и аспектах аналогичных использованным в языке GRAMMATIC.

Механизм шаблонов GRAMMATIC обобщается для произвольного языка следующим образом. Выделяется набор *базовых понятий* (синтаксических категорий или классов в целевой мета-модели), характеризующих такой механизм в любом языке: *шаблон*, *параметр*, *шаблонное выражение*, *применение шаблона*, *использование шаблонного параметра*. Для каждого конкретного языка этот набор дополняется конкретными типами шаблонных выражений, соответствующих конструкциям данного языка. Шаблонные версии конструкций допускают использование шаблонных параметров и применений шаблонов вместо тех или иных своих частей.

Семантика шаблонов определяется алгоритмом разворачивания, который обозначается $\mathcal{I}_\gamma[\bullet]$, где *среда* γ сопоставляет шаблонным параметрам фактические значения. Для применений базовых понятий этот алгоритм работает одинаково вне зависимости от конкретного языка, а для специфичных шаблонных выражений создаются экземпляры соответствующих конструкций языка и алгоритм вызывается рекурсивно на подвыражениях. Конечность такой процедуры обеспечивается тем, что шаблонам

запрещено вызывать себя рекурсивно, и результирующие объекты, таким образом, всегда конечны. Время работы алгоритма разворачивания линейно от размера входных данных.

Для того, чтобы диагностика ошибок происходила в терминах исходной спецификации, содержащей шаблоны, а не в терминах результата разворачивания, необходимо иметь возможность убедиться в корректности результата, не осуществляя разворачивания. Эта задача решается с помощью системы типов, которая строится следующим образом: правила типизации для базовых понятий фиксированы; наиболее важные правила приведены на Рис. 1. Для специфичных шаблонных выражений генери-

$$\frac{}{\Gamma \cup \{v : \tau\} \vdash <?v> : \tau}\ \text{var}$$

$$\frac{\Gamma \cup \bigcup_{i=1}^{n} \{p_i : \tau_i\} \vdash b : \sigma}{\Gamma \vdash \mathbf{template}\,(T<p_1 : \tau_1, \ldots, p_n : \tau_n> : \sigma\,\{b\})}\ \text{abstr}$$

$$\frac{\Gamma \vdash \mathbf{template}\,(T<p_1 : \tau_1, \ldots, p_n : \tau_n> : \sigma\,\{b\}) \quad \forall i : [1:n].\ \Gamma \vdash a_i : \tau_i}{\Gamma \vdash <T\ a_1, \ldots, a_n> : \sigma}\ \text{appl}$$

Рис. 1: Основные правила системы типов языка шаблонов

руются новые правила системы типов, отражающие структурные ограничения, накладываемые целевой мета-моделью исходного языка (включая кратность ссылок).

Для построенных таким образом алгоритма $\mathcal{I}_\gamma[\bullet]$ и системы типов доказаны следующие теоремы, показывающие, что система типов гарантирует корректность результата разворачивания.

Теорема 4.1 (О сохранении типов)**.** *Если среда γ согласована с контекстом Γ и $\Gamma \vdash e : \tau$, то $\Gamma \vdash \mathcal{I}_\gamma[e] : \tau$. Другими словами, преобразование $\mathcal{I}[\bullet]$ сохраняет типы.*

Теорема 4.2 (О нормализации)**.** *Если среда γ согласована с контекстом Γ и $\Gamma \vdash e : \tau$, то вычисление $\mathcal{I}_\gamma[e]$ заканчивается за конечное количество шагов, и результат не содержит применений шаблонов и шаблонных параметров.*

Аналогично описанному выше строится алгоритм реконструкции типов, позволяющий пользователю не декларировать типы шаблонов и их параметров явно.

	Lisp	MacroML	Haskel TMP	cpp	Velocity и др.	MPS	Reuseware	*Данная работа*
Поддержка внешних ПОЯ	-	-	-	+	+	+	+	+
Применимо для текстовых языков	n/a	n/a	n/a	+	+	-	+	+
Замкнутость	+	+	+	+	+	n/a	-	+
Контроль корректности	n/a	+	+	-	-	+	+	+
Логика в теле шаблона	+	+	+	+/-	+	+	-	-

Таблица 3: Поддержка шаблонов

Поддержка аспектов строится на тех же принципах, что и поддержка шаблонов, причем многие конструкции, генерируемые для поддержки шаблонов используются также и для аспектов. Основные отличия лежат в области семантики, которая в случае аспектов описывается двумя алгоритмами: сопоставление выражения e исходного языка с образцом (срезом) p, обозначаемое e **match** p, и применение аспектного правила $\mathcal{R}$ в случае успешного сопоставления, обозначаемое $\mathcal{R}@e$.

Операция сопоставления является в некотором смысле обратной к разворачиванию шаблона: вместо того, чтобы заменять вхождения переменных их значениями, она должна найти значения переменных, удовлетворяющие данному образцу. Основной интерес представляет алгоритм сопоставления коллекций (списков и множеств) в случае использования подстановочных знаков, допускающих сопоставление с частью коллекции. В случае списков для решения этой задачи применяется перебор с возвратами, а в случае множеств задача сводится к поиску максимальных паросочетаний в друдольных графах. В литературе показано, что задача сопоставления образцов является NP-полной при наличии переменных и неупорядоченных коллекций, и используемый алгоритм имеет экспоненциальную оценку времени работы в худшем случае, однако этот алгоритм показывает вполне приемлемые результаты на практике, что было установлено в процессе использования аспектов, реализованных в Grammatic.

Применение аспектного правила сводится к замене объектов, сопоставленных переменным, другими объектами, полученными при разворачивании соответствующих шаблонов. В общем случае смысл имеют только советы с ключевым словом `instead`, то есть заменяющие, а не добавляющие, поскольку понятия вставки до и после могут не иметь смысла в кон-

кретном языке. Тем не менее, если эти понятия имеют смысл, они легко реализуются через замену.

Система типов, используемая для статического контроля корректности результатов применения аспектов базируется на системе типов для шаблонов с той лишь разницей, что в случае аспектов требуется соответствие между типом переменной и шаблонным выражением, которым значения этой переменной будут заменены, что выражается следующим правилом:

$$\frac{\mathcal{R} = \langle p, T, V \rangle \quad p \vdash v : \tau \quad \Gamma(p) \vdash t : \sigma \quad \sigma \preceq \tau}{(\textbf{instead}\ v\ :\ t) \in \mathit{Allowed}(\mathcal{R})}\ \mathit{aspect}$$

Здесь $\mathit{Allowed}(\mathcal{R})$ обозначает множество всех корректных пар “переменная-шаблонное выражение”.

	JAMI	AspectJ	XCPL	Reuseware	Aspect.NET	POPART	XAspects	Van Wyk'03	Stratego/XT	*Данная работа*
Поддержка многих языков	+	-	+	+	+	+	+	+	+	+
Независимость от платф.	-	-	+	+	-	+	-	+	+	+
Неисполняемые языки	-	-	+	+	-	-	-	+	+	+
Автоматизация	n/a	-	-	+	+	-	+	+	-	+
Незнание	+	+	+	-	+	+	+	+	+	+
Спец. язык срезов	-	+	+	-	+	+	+	-	+	+
Семантика встраивания	+	+	-	+	+	+	+	+	+	+

Таблица 4: Поддержка аспектов

В качестве примера применения описанного метода разработано расширение языка описания мета-моделей Emfatic, добавляющее в данный язык поддержку композиции с помощью шаблонов и аспектов.

Основные результаты диссертационной работы

- Разработан предметно-ориентированный язык описания текстового синтаксиса, реализующий механизмы композиции, основанный на использовании модулей, шаблонов и аспектов. Продемонстрирована эффективность механизмов композиции, использованных в данном языке.
- Разработан генератор трансляторов, поддерживающий проверку типов в семантических действиях и гарантирующий отсутствие ошибок типизации в сгенерированном коде сразу для многих языков реализации.
- Предложен метод автоматизации разработки предметно-ориентированных языков с поддержкой механизмов композиции, основанных на шаблонах и аспектов.
- С использованием предложенного подхода созданы расширения языка Emfatic.

ПУБЛИКАЦИИ ПО ТЕМЕ ДИССЕРТАЦИОННОЙ РАБОТЫ

Журналы, входящие в перечень ВАК

1. *Бреслав, А. А.* Средства повторного использования формальных грамматик и их применение для создания диалектов / А. А. Бреслав // *Научно-технический вестник СПбГУ ИТМО.* — Выпуск №61. — 2009. — С. 75–81.
2. *Бреслав, А. А.* Применение принципов MDD и аспектно-ориентированного программирования к разработке ПО, связанного с формальными грамматиками / А. А. Бреслав, И. Ю. Попов // *Научно-технический вестник СПбГУ ИТМО.* — Выпуск №57. — 2008. — С. 87–96.

Прочие публикации

1. *Breslav, A.* Creating textual language dialects using aspect-like techniques / A. Breslav // Pre-proceedings of 3rd Summer School on Generative and Transformational Techniques in Software Engineering (Braga, Portugal). — 2009. — Pp. 63–64.

2. *Breslav, A.* A type checker for multi-targeted parser specifications / A. Breslav // Proceedings of 15th Estonian Winter School in Computer Science (Palmse, Estonia). — 2010. — Pp. 12–13.

3. *Бреслав, А. А.* Создание диалектов языков программирования с использованием грамматических аспектов / А. А. Бреслав // Компьютерные науки и информационные технологии: Материалы Междунар. науч. конф. / Под ред. В. А. Твердохлебов, Ю. И. Митрованов, А. Г. Федорова и др. — Саратов: Издательство Саратовского университета, 2009. — С. 231–232.