Программное обеспечение синтеза SQL запросов на основе метамодели инструментария BlockSet

Н.А. Козырев, П.П. Кейно

***Аннотация*—В работе рассматривается одна из основных задач при разработке интерпретатора для инструментария BlockSet – проектирование синтеза SQL запросов на основе метамодели, а так программного обеспечения для его применения. Авторами рассмотрен как инструментарий в целом, так и роль самого алгоритма в его контексте. Подробно рассмотрены периферийные инструменты для управления алгоритмом в разках языка BML. Продемонстрированы этапы его формирования и технические тонкости реализации. Помимо прямого алгоритма так же рассмотрен обратный алгоритм генерации запроса для поиска факторов частных событий, который необходим при реализации ресурса на основе событийно-ориентированного подхода. Для наглядности разобран пример обработки события появления "сообщения", в результате которого необходимо определять id "пользователей" -- отправителя и получателя, которых нужно уведомить о появлении указанного "сообщения". Подведены выводы, демонстрирующие, результаты проделанной работы.**

***Ключевые слова*—BlockSet, BML, алгоритм, синтез, метамодель, инструментарий, SQL, Web, предметно-ориентированное программирование, DSL.**

I. Введение

Информационные системы на основе Web-технологий охватывают широкий спектр задач, поскольку гипертекстовая среда стала доминантой сети Интернет. Многократно растёт и количество небольших проектов, включающих динамические Web-приложения. Тенденции развития современной Web-разработки заключаются не только в увеличении числа типовых решений, но и всё большей востребованности приложений, разработанных по индивидуальному проекту.

Использование языков программирования общего назначения для построения логики серверной части имеет большой порог вхождения, а традиционный CMS и ЯП фреймворки не обеспечивают достаточную гибкость. Наиболее верным решением в такой ситуации будет использовать DSL-подобные языки [1], которые просты в обращении и при этом имеют обширные возможности благодаря тому, что заранее ориентированы на решение конкретных задач. Однако на данный момент существует очень мало готовых решений. При исследовании существующих решений наиболее близким к BlockSet'у по функционалу оказался WebDSL [2], однако он имеет ряд недостатков, унаследованных от языка реализации данного проекта Java, таких как большие затраты производительности и необходимость устанавливать Java-машину.

Проект BlockSet отличается от конкурентов, решая описанные выше задачи без потерь гибкости и производительности и предоставляет инструменты для реализации каждого из этапов создания динамического Web-ресурса. Одним из таких инструментов является декларативный, высокоуровневый, предметно-ориентированный язык BML [3-5], созданный на основе семантически-нейтрального XML. Данный язык служит для описания логики взаимодействия пользователя, а точнее пользовательских запросов, и базы данных ресурса. Именно о реализации алгоритма синтеза SQL запросов на основе конструкций языка BML пойдёт речь в данной работе.

II. Язык BML

BML (BlockSet Modelling Language) – это декларативный, предметно-ориентированный высокоабстрактный язык, созданный на основе семантически нейтрального XML. Он является одним из основных инструментов BlockSet. Он необходим для описания устройства всего web-ресурса и логики бэк-энда. BML оперирует четырьмя тэгами:

- Block (блок). Каждый тэг block описывает один из фрагментов данных, которыми оперирует web-ресурс Есть несколько типов блоков, каждый из который соответствует какому-либо типу данных, причём важно понимать, что для сложных типов блока его представление может состоять из нескольких параметров, например для блока типа file актуальны такие параметры как имя и url
- Set (Набор, сет). Каждый set является представлением некоторой абстракции и состоит из блоков. Хорошим примером сета является такая абстракция как пользователь. В таком случае этот сет может состоять из таких блоков как ФИО, возраст, логин, пароль, аватар.
- Model (Модель). Модель описывает общую архитектуру данных web-ресурса
- Location (Локация). Локаций может быть несколько. Каждая из них описывает отдельный метод web-

Статья получена 10 марта 2024.

Н. А. Козырев, аспирант Московского Авиационного Института, Москва, Россия (e-mail: science@blockset.ru).

П. П. Кейно, доцент кафедры №316 Московского Авиационного Института, Москва, Россия (e-mail: science@blockset.ru).

ресурса. Содержит дерево сетов, которое интерпретируется как последовательность выборки и реализует логику бэк-энда для данной локации.

Дерево вложенных сетов в локации описывает алгоритм, задаваемый при её вызове. И ключевым оператором при написании такого декларативного алгоритма является размещение сета внутри другого сета. Такое вложение описывает необходимость их связи при синтезе алгоритма выборки, однако эта связь может быть реализована разным образом. Для определения типа связи предусмотрен атрибут nest, который может принимать следующие значения:

- inherit
- trim
- attach

По умолчанию атрибут имеет значение inherit, которое и показывает, что выборка внутреннего сета будет унаследована от внешней, т.е. будут выбраны только те экземпляры сета, который соответствуют экземплярам внешнего, однако для указанного атрибута так же можно указать значение trim, которое наделяет связь сетов двунаправленностью, что укажет необходимость для выборки экземпляров внешнего сета и может зависеть от выборки внутреннего. Значение attach не налагает никаких ограничений ни на один из сетов данной связи, но указывает, что после полного формирования внешней и внутренней выборок они должны приобрести соответствие друг-другу. Это полезно, например, при создании новых экземпляров сета или обеспечения связи уже существовавших.

III. Разработка алгоритма формирования SQL-запросов

Алгоритм формирования SQL-запросов основан на итеративном обходе дерева сета в управляющей локации от корня к ветвям. При обходе приоритет отдаётся ограничивающим сетам, при этом для них используется рекурсия с предобработкой, что позволяет в первую очередь учесть ограничения, расположенных наиболее глубоко во вложенной цепочке в ситуации наличия нескольких вложенных друг в друга ограничивающих сетов. На схеме ниже представлены зависимости сетов с разными видами вложения. Логика, описанная в сете, будет выполнятся только после того, как удовлетворены все зависимости, т.е. выполнена логика сетов, векторы от которых ведут к текущему. Стоит отметить, что сеты, вложенные с использованием связи trim будет необходимо вычислить дважды – первично для определения собственных ограничений и вторично – для наследования ограничений выборки экземпляров родительского сета (рис. 1).

При такой реализации ограничения в сете влияют на все зависимые от него сеты, а их количество может быть больше одного, что означает, что для достижения наилучшей производительности необходимо кэшировать результаты работы SQL-запроса на ограничивающем сете, однако если выполнять кэширование на уровне кода, то каждый SQL-запрос придётся вызывать отдельно, что тоже негативно сказывается на производительности.

Обработка сетов со значением attach атрибута nest отличается от продемонстрированной логики. В случае её наличия выбор ребра зависимости исходит из типа связи сета с вышестоящим – один ко многим, многие к одному или многие ко многим. Такая логика обусловлена особенностями устройства связей таблиц в базе данных. При связи один ко многим внешний ключ, который и подлежит изменению при соединении экземпляров сетов, находится в вышестоящем сете, поэтому его необходимо обрабатывать в последнюю очередь. Для связи многие к одному ситуация противоположная. Для связи многие ко многим направление ребра зависимости значения не имеет, т.к. изменения связей кортежей будут производиться в таблице связанности и не затрагивают в логике формирования конструкций WITH ни одного из связанных сетов.

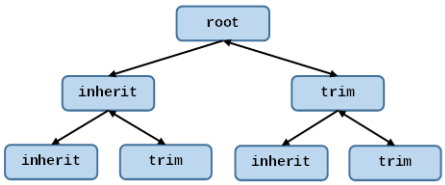

Рис. 1. Организация выборки данных в сетах

Во избежание выполнения SQL-запросов по отдельности был изобретён алгоритм, позволяющий кэшировать значения отдельных подзапросов непосредственно во время выполнения общего запроса. Этот алгоритм заключается в применении и комбинировании конструкций WITH, встроенным в СУБД PostgreSQL [6-8]. Эта конструкция позволяет выполнять отдельные SQL-запросы перед основным и использовать результаты, полученные в них как отдельную именованную таблицу, причём в процессе выполнения общего запроса выгруженные данные хранятся в оперативной памяти, что и реализует функционал кэширования. Реализация такого алгоритма напрямую связана с представленной выше схемой зависимостей – WITH'ы вызываются в таком порядке, чтобы все зависимости были удовлетворены. Далее, после обработки всего дерева сетов таким образом, происходит вызов основного запроса, задачей которого является выбрать результаты работы вышестоящих подзапросов, которые были отмечены для чтения. Вариант соединять результаты с использованием классической конструкции JOIN был быстро отвергнут, т.к. это приводило к возникновению большого количества дубликатов, поэтому для выгрузки результатов работы SQL-запроса был изобретён формат выбора лестницей. Такой формат позволяет уместить в сразу несколько таблиц с собственными атрибутами в рамках одной. Имена атрибутов при этом генерируются таким образом, чтобы при выгрузке по типу строки и идентификации сета, к которому она относится, была возможность воспроизвести его имя атрибута и сразу получить доступ к искомой ячейке в строке (рис. 2).

Рис. 2. Схема получения данных из нескольких таблиц

В дополнение к выгружаемым данным для каждой строки дополнительно выгружаются идентификаторы строк дочернего сета для сохранения связанности данных При чтении результатов эти идентификаторы сразу сохраняются как ключи хэш-таблиц.

Далее для каждой выгруженной строки происходит проверка прав доступа, алгоритм которой подробнее описан в работе [9]. Далее происходит формирования ответа в формате JSON. Выгруженные строки иерархически вкладываются друг в друга в соответствии с форматом, описанным в атрибуте fetch у каждого сета. Воспроизведение связанности данных в структуре JSON [10], а именно поиск вложенных строк для каждой строки внешнего сета осуществляется с использованием хэш-функции по искомому идентификатору, что обеспечивает производительность, близкую к **O(1)** для каждой строки, т.е. **O(n)** для всего запроса, где **n** – это количество выгружаемых строк всех сетов.

IV. Разработка алгоритма обработки событий

Поддержка событийно-ориентированного подхода реализована путём подписки пользователя на отдельные события.

При совершении какого-либо действия с экземпляром сета (создание, изменение или удаление) формируется многозначная хэш-таблица [11], ключами в которой являются идентификаторы событий, состоящие из имени сета и имени действия, а значениями – id экземпляров в рамках сета, с которыми было произведено означенное действие. Далее для каждого действия обработчик подписок производит поиск среди всех подписок, в результате которого выбираются только отслеживающие совершённое действие. Далее для каждого отобранного слушателя по имени действия производится поиск сета, доступного к прослушиванию. Для каждого сета выполняется процесс индукции, который необходим для вычисления комбинаций значений плейсхолдеров URI локации, которые приводят к возникновению экземпляра с идентификатором из списка идентификаторов для актуального события в выборке на прослушиваемом сете. Для большего удобства этот процесс будет разобран на примере локации личных сообщений. Локация имеет два плейсхолдера: для идентификатора пользователя текущей сессии и для идентификатора пользователя, диалог с которым открыт. Корневой сет представляет множество сообщений, которые должны присутствовать на такой странице и состоит из отправленных и полученных сообщений, что реализовано двумя сетами со значением trim атрибута nest и значением or атрибута join на корневом сете, который задаёт дизъюнкцию как метод объединения ограничивающих сетов. Сет отправленных сообщений реализован с помощью двух вложенных ограничивающих сетов пользователей – отправителя и получателя, с именами, соответствующими плейсхолдерам пользователя сессии и другого пользователя соответственно. Сет полученных сообщений реализован аналогично, но в этом случае пользователь сессии прикреплён к плейсхолдеру получателя, а другой пользователь – к плейсхолдеру отправителя. Так же для корневого сета предусмотрены блоки и сеты для получения информации о тексте сообщения, отправителе и получателе (рис 3).

```
<location base="/[logined_user@id]/dialogs/[another_user@id]" >
  <set id="messages" act="read" join="or" name="messages" listen="create" >

    <set id="@parent" act="read" nest="trim" fetch="none" name="sent_messages" >
      <set id="_users" by="author" act="read" nest="trim" name="logined_user">
      </set>
      <set id="_users" by="reciever" act="read" nest="trim" name="another_user">
      </set>
    </set>

    <set id="@parent" act="read" nest="trim" fetch="none" name="recieved_messages" >
      <set id="_users" by="author" act="read" nest="trim" name="another_user">
      </set>
      <set id="_users" by="reciever" act="read" nest="trim" name="logined_user">
      </set>
    </set>

    <block name="text" />
    <set id="_users" by="author" act="read" name="author">
      <block name="_login" /> </set>
    <set id="_users" by="reciever" act="read" name="reciever">
      <block name="_login" /> </set>

  </set>
</location>
```

Рис. 3. Исходный код BML с демонстрацией событийно-ориентированного подхода

В основе индуктивного алгоритма лежит рекурсивный обход дерева сетов особым образом, начиная с прослушиваемого сета. Так как задачей алгоритма является поиск переменных, приведших к определённому событию, для его выполнения необходимо обратить причинно-следственную связь. Для достижения такого результата на каждой итерации рекурсии обход продолжается только в сторону сетов, ребро зависимости от которых направлено к текущему, т.е. в родительский сет при его наличии и во все дочерние сеты со значением trim атрибута nest. На каждой итерации рекурсии производится как предобработка, производимая перед дальнейшим углублением, так и пост-обработка, производимая после завершения вложенных вызовов рекурсии. На рисунке ниже (рис 4) отмечена структура дерева, по которому произведётся обход. Чёрными стрелками отмечены рёбра зависимости, красными – путь обхода (Рис 4).

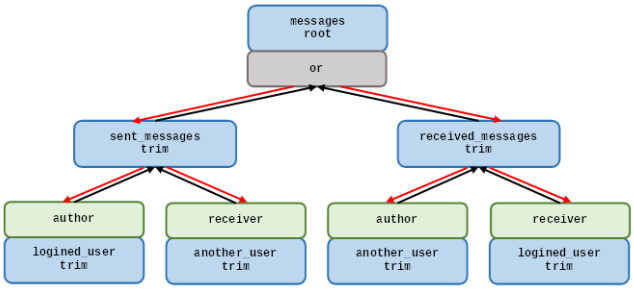

Рис. 4. Структура дерева путей обхода

Целью предобработки является первичное определение возможных значений плейсхолдеров URI на каждом обходимом сете. Для каждого сета определяется сет его экземпляров. Для прослушиваемого сета критерием выборки является идентификатор, полученный на этапе формирования события. Для остальных сетов критерием для выбора каждого экземпляра является его связь одним из экземпляров выборки сета на верхнем уровне рекурсии. Помимо первичных и внешних ключей для каждого экземпляра сета генерируются значения каждого плейсхолдера. Для плейсхолдеров, связанных с данным сетом проставляются соответствующие значения (идентификатор или значение блока), для остальных оставляется NULL, что показывает отсутствие ограничений данного плейсхолдера на этом сете. Для рассматриваемого примера возьмём ситуацию, в которой пользователь с идентификатором 1 отправил пользователю с идентификатором 2 сообщение с идентификатором 10. Ниже представлена таблица с идентификаторами выбранных экземпляров сетов и сгенерированными для них возможными значениями плейсхолдеров. Экземпляров может быть выбрано несколько, однако для нашего примера такой вариант не актуален. Зелёным цветом отмечены совпадения имени плейсхолдера и сета, что и является критерием их связи для разбираемого случая (рис. 5).

	идентификатор	logined_user	another_user
messages	10	null	null
sent_messages	10	null	null
logined_user (author)	1	1	null
another_user (receiver)	2	null	2
received_messages	10	null	null
another_user (author)	1	null	1
logined_user (reciever)	2	2	null

Рис. 5. Таблица с идентификаторами выбранных экземпляров сетов

Целью постобработки на каждой итерации рекурсии является соединение всех ограничений плейсхолдеров с более нижнего уровня рекурсии, а также собственных ограничений, результатом является итоговые ограничения сета, которые в дальнейшем будут использованы на верхнем уровне рекурсии. Такой результат на самом верхнем уровне и будет являться искомым списком возможных значений плейсхолдеров. В процессе соединения используются два алгоритма – AND и OR, и три основные группы ограничений: собственные ограничения сета, ограничения родительского сета с нижнего уровня рекурсии и ограничения дочерних сетов с нижних уровней рекурсии. Эти три части всегда объединяются по алгоритму AND, однако ограничения дочерних сетов с нижних уровней рекурсии сами по себе требуют объединения и в этом случае алгоритм выбирается на основе значения атрибута JOIN актуального сета. Принцип работы алгоритма OR максимально прост. В процессе его выполнения происходит простое склеивание ограничений, однако алгоритм AND работает по более сложным правилам. Для двух списков ограничений происходит сравнение каждого ограничения из первого списка с каждым из второго. Для каждой пары ограничений сравниваются их значения для каждого плейсхолдера. Сравнение значений для плейсхолдера считается успешным в случае, если они равны, либо, в случае если хотя бы один из равен NULL. Результирующим значением плейсхолдера считается либо существующее значение (любое, отличное от NULL), либо NULL, в случае отсутствия таковых в сравнении. В случае успешного сравнения значений для всех плейсхолдеров результатом сравнения двух ограничений является ограничение, состоящее из результирующих значений плейсхолдеров, в противном случае сравнение признаются неуспешным и результат отсутствует. Результатом соединения двух списков ограничений является список результатов попарного сравнения ограничений.

В табл. (рис 6) представлена работа постобработки на каждой итерации рекурсии. Таким образом, результатом разобранного примера являются два варианта ограничений – 1, 2 и 2, 1, которые как раз и являются искомыми значениями плейсхолдеров локаций двух пользователей, открывших диалог друг с другом.

				logined_user	another_user
messages	Итоговый результат			1	2
				2	1
	Собственные ограничения				
	Соединение дочерних (OR)			1	2
				2	1
	sent_messages	Итоговый результат		1	2
		Собственные ограничения			
		Соединение дочерних (AND)		1	2
		logined_user (author)	Итоговый результат	1	null
			Собственные ограничения	1	null
			Соединение дочерних		
		another_user (receiver)	Итоговый результат	null	2
			Собственные ограничения	null	2
			Соединение дочерних		
	recieved_messages	Итоговый результат		2	1
		Собственные ограничения			
		Соединение дочерних (AND)		2	1
		another_user (author)	Итоговый результат	null	1
			Собственные ограничения	null	1
			Соединение дочерних		
		logined_user (receiver)	Итоговый результат	2	null
			Собственные ограничения	2	null
			Соединение дочерних		

Рис 1. Таблица постобработки на каждом этапе рекурсии

Реализуется индуктивный алгоритм на уровне базы данных в рамках единого SQL-запроса, что обеспечивает его высокое быстродействие. Для поэтапного выполнения рекурсии с поддержкой переиспользования результатов используется конструкция WITH, для реализации алгоритма OR – конструкция UNION, для реализации алгоритма AND – соединение INNER JOIN для попарного сравнения ограничений с соответствующими критериями сравнения и конструкция COALESCE для приоритезации результирующих, значений сравнения.

V. Заключение

Разработан гибкий алгоритм для синтеза SQL запросов на основе метамодели инструментария BlockSet, а также программное обеспечения для его. В данной работе рассматривается одна из основных задач при разработке интерпретатора для инструментария BlockSet – проектирование синтеза SQL запросов на основе метамодели, а так программного обеспечения для его применения. Подробно рассмотрены периферийные инструменты для управления алгоритмом в разках языка BML.

Продемонстрированы этапы его формирования и технические тонкости реализации. Помимо прямого алгоритма так же рассмотрен обратный алгоритм генерации запроса для поиска факторов частных событий, который необходим при реализации ресурса на основе событийно-ориентированного подхода. Для наглядности разобран пример обработки события появления “сообщения”, в результате которого необходимо определять id “пользователей” - отправителя и получателя, которых нужно уведомить о появлении указанного “сообщения”.

Подведены выводы, демонстрирующие, результаты проделанной работы.

Библиография

[1] Cadavid J. J. et al. A Domain Specific Language to Generate Web Applications// In *CIbSEm*, 2009, pp. 139-144.

[2] Visser E. *WebDSL. A Case Study in Domain-Specific Language*. Berlin, Heidelberg. Springer 2008

[3] Кейно П.П. Предпосылки формирования новой методологии разработки веб-узлов BlockSet и декларативного языка BML // *Современные информационные технологии и ИТ-образование*, т. 11б № 2, с. 78-84, 2015..

[4] Кейно П.П. Разработка архитектуры программного комплекса синхронизатора при интерпретаторе декларативного языка BML // *Прикладная информатика*, т. 11, № 2 (62), с. 65-77, 2016.

[5] Hanus M., Koschinicke S. An ER-based framework for declarative web programming // In *Practical Aspects of Declarative Languages*, 2010, pp. 201-216

[6] Боровский И.Г., Шельмина Е.А. Сравнительный анализ настольных и клиент-серверных СУБД // *Доклады Томского государственного университета систем управления и радиоэлектроники*, т, 20, №4, с. 92-94, 2017.

[7] The PostgreSQL Global Development Group. PostgreSQL documentation: Composites. URL: http://www.postgreSQL.org/docs/10/rowtypes.html (дата обращения 24.02.2024)

[8] The PostgreSQL Global Development Group. PostgreSQL documentation: Arrays. URL: http://www.postgreSQL.org/docs/10/arrays.html (дата обращения 24.02.2024)

[9] Кейно П. П., Козырев Н.А., Квашнин В.М., Новиков А.Ю. Организация системы прав доступа и ее управления в проекте BlockSet // Прикладная информатика, т. 14, № 3(81), с. 66-73, 2019.

[10] Nicolas Seriot. Parsing JSON is a Minefield. URL: htpp://http://seriot.ch/parsing_JSON.php (дата обращения 24.04.2019)

[11] Truica C. O. et al. Performance evaluation for CRUD operations in asynchronously replicated document oriented database // In *2015 20th International Conference on Control Systems and Computer Science (CSCS)*,. IEEE, 2015, p. 191-196..

[12] Johanson A.N., Hasselbring W. Effectiveness and efficiency of a domain-specific language for high-performance marine ecosystem simulation: a controlled experiment // *Empirical Software Engineering*, vol. 22, pp. 2206-2236, 2017.

[13] Karus S., Dumas M. Predicting the maintainability of XSL transformations // *Science of Computer Programming*, vol. 76, no. 12, pp. 1161–1176, 2011.

[14] Kleene S.C. *Representation of events in nerve nets and finite automata*: tech. rep. DTIC Document. 1951.

[15] Klint P., Van Der Storm T., Vinju J. On the impact of DSL tools on the maintainability of language implementations // In *Proceedings of the Tenth Workshop on Language Descriptions, Tools and Applications*, ACM. 2010, p. 10.

[16] Kosar T., Oliveira N., Mernik M., Pereira V. J. M., Crepinsˇek M., Da C. D., Henriques R. P. Comparing general-purpose and domain-specific languages: An empirical study // *Computer Science and Information Systems*, vol. 7, no. 2, pp. 247–264, 2010.

[17] Li C. T., El Gamal A. An efficient feedback coding scheme with low error probability for discrete memoryless channels // *IEEE Transactions on Information Theory*, vol. 61, no. 6, pp. 2953–2963, 2015.

[18] Magel K., Kluczny R.M., Harrison W.A., Dekock A.R. *Applying software complexity metrics to program maintenance*. 1982.

SQL query synthesis software based on the metamodel of the BlockSet toolkit

Nikolay Kozyrev, Pavel Keyno

***Abstract*— The paper examines one of the main tasks in developing an interpreter for the BlockSet toolkit - designing the synthesis of SQL queries based on the metamodel, as well as software for its use. The authors examined both the toolkit as a whole and the role of the algorithm itself in its context. Peripheral tools for managing the algorithm in the BML language are discussed in detail. The stages of its formation and technical details of implementation are demonstrated. In addition to the direct algorithm, a reverse algorithm for generating a query to search for factors of private events, which is necessary when implementing a resource based on an event-oriented approach, is also considered. For clarity, an example of processing the "message" event has been analyzed, as a result of which it is necessary to determine the id of "users" - the sender and the recipient, who need to be notified about the appearance of the specified "message". Conclusions are drawn demonstrating the results of the work done.**

***Keywords*—BlockSet, BML, algorythm, synthesis, metamodel, toolkit, SQL, Web, domain-specific programming, DSL.**

REFERENCES

[1] J.J. Cadavid et al., "A Domain Specific Language to Generate Web Applications," In *CIbSE*, 2009, pp. 139-144.

[2] E. Visser, *WebDSL. A Case Study in Domain-Specific Language*. Berlin, Heidelberg. Springer 2008.

[3] P.P. Keyno, "Predposylki formirovaniya novoj metodologii razrabotki veb-uzlov BlockSet i deklarativnogo yazyka BML," [Prerequisites for the formation of a new methodology for developing BlockSet web sites and the declarative language BML]. *Sovremennye informacionnye tekhnologii i IT-obrazovanie*. [Modern information technologies and IT education.], vol. 11, no. 2, pp. 78-84, 2015. [in Rus]

[4] P Keyno, "Software Architecture of synchronizer as part of declarative web-application modeling language interpreter," *Prikladnaya Informatika* [*Journal of Applied Informatics*], vol. 11, no. 2 (62), pp. 65–77, 2016. [In Rus].

[5] M. Hanus, S. Koschinicke, "An ER-based framework for declarative web programming," In *Practical Aspects of Declarative Languages*, 2010, pp. 201-216

[6] I.G. Borovskoy, E.A Shelmina, "Comparative analysis of desktop and client-server databases," *Doklady Tomskogo gosudarstvennogo universiteta sistem upravlenija i radiojelektroniki*, vol. 20, no 4, pp. 92-94, 2017.

[7] The PostgreSQL Global Development Group. PostgreSQL documentation: Composites. URL: http://www.postgreSQL.org/docs/10/rowtypes.html

[8] The PostgreSQL Global Development Group. PostgreSQL documentation: Arrays. URL: http://www.postgreSQL.org/docs/10/arrays.html

[9] P. Keyno, N. Kozyrev, V. Kvashnin, A. Novikov, "Permission system and its management organization in BlockSet project," *Prikladnaya informatika* [Journal of Applied Informatics], vol. 14, no. 3 (81), pp. 66-73, 2019. [In Rus]

[10] Nicolas Seriot. Parsing JSON is a Minefield. URL: htpp://http://seriot.ch/parsing_JSON.php

[11] C.O. Truica et al, "Performance evaluation for CRUD operations in asynchronously replicated document oriented database," In *2015 20th International Conference on Control Systems and Computer Science* (CSCS),.IEEE, 2015, pp. 191-196.

[12] A.N., Johanson, W. Hasselbring, "Effectiveness and efficiency of a domain-specific language for high-performance marine ecosystem simulation: a controlled experiment," *Empirical Software Engineering*, vol. 22, pp. 2206-2236, 2017.

[13] S. Karus, M. Dumas, "Predicting the maintainability of XSL transformations," *Science of Computer Programming*, vol. 76, no. 12, pp. 1161–1176, 2011.

[14] S.C. Kleene, *Representation of events in nerve nets and finite automata*: tech. rep. DTIC Document. — 1951.

[15] P. Klint, T. Van Der Storm, J. Vinju, "On the impact of DSL tools on the maintainability of language implementations," In *Proceedings of the Tenth Workshop on Language Descriptions, Tools and Applications*, ACM. 2010, p. 10.

[16] T. Kosar, N. Oliveira, M. Mernik, V.J.M. Pereira, M. Crepinsˇek, C.D. Da, R.P. Henriques, "Comparing general-purpose and domain-specific languages: An empirical study," *Computer Science and Information Systems*, vol. 7, no. 2, pp. 247–264, 2010.

[17] C.T. Li, A. El Gamal, "An efficient feedback coding scheme with low error probability for discrete memoryless channels," *IEEE Transactions on Information Theory*, vol. 61, no. 6, pp. 2953–2963, 2015.

[18] K. Magel, R.M. Kluczny, W.A. Harrison, A.R. Dekock, *Applying software complexity metrics to program maintenance*. 1982.

About of Authors

Nikolay A. Kozyrev, postgraduate student, Moscow Aviation Institute, Moscow, Russia (e-mail: science@blockset.ru).

Pavel P. Keyno, Associate Professor, Department No. 316, Moscow Aviation Institute, Moscow, Russia (e-mail: science@blockset.ru).