Потоковая обработка больших документов в ситуационно-ориентированных базах данных

А. С. Гусаренко
Уфимский государственный авиационный технический университет
e-mail: gusarenko@ugatu.su

В. В. Миронов
Уфимский государственный авиационный технический университет
e-mail: mironov@ugatu.su

Н. И. Юсупова
Уфимский государственный авиационный технический университет
e-mail: yussupova@ugatu.ac.ru

Аннотация

В статье обсуждается потоковая обработка данных в больших объемах для ситуационно-ориентированных баз данных (СОБД). Традиционно в СОБД предусмотрена кэшированная обработка разнородных данных. Рассматривается работа с файлами большого объема, не помещающихся целиком в оперативную память, – содержащих множество однотипных фрагментов, которые могут обрабатываться порциями. Порции данных извлекаются из входного потока, обрабатываются в буфере, после чего направляются в выходной поток. Рассматриваются три варианта реализации данной схемы в рамках иерархической ситуационной модели HSM. Предлагаемые на концептуальном уровне средства иллюстрируются на примере обработки XML-данных с использованием программных инструментов PHP, таких как XMLReader, XMLWriter, DOM.

Ключевые слова: динамическая модель, модель СОБД, базы данных, потоковая обработка, кэшированная обработка, обработка данных, DOM, HSM.

1. Введение

На современном этапе развития информационных систем, доступных через сеть, в их архитектуре используются большие массивы данных, накапливаемые в процессе их жизненного цикла. В составе таких крупномасштабных информационных систем в качестве информационного обеспечения используются базы данных, помогающие справиться с задачами обработки, хранения данных и сопровождения автоматизированных функций [1, 2]. В последнее время выделяются два направления развития СУБД, первое SQL базы данных и NoSQL, получившее в последнее время заслуженное внимание за счет использования не только SQL-средств, но и своих собственных структур данных в форме документов, пригодных для обработки и хранения, а также языковых средств, не использующих SQL.

Ситуационно-ориентированные базы данных (СОБД) являются интегратором разнородных (гетерогенных) данных. В составе СОБД формализована динамическая модель HSM [3] и запрограммирована функциональность для обработки текущих состояний приложения, базирующегося на возможностях СОБД. В состояниях модели обрабатываются данные из таблиц реляционной СУБД, таких как MySQL [4, 5], к которым они привязаны. На начальном этапе это были технологии кэшированной обработки [6] в основу технологии была положена DOM (Document Object Model – объектная модель документа), используемая во множестве современных приложений, ориентированных на веб-приложения и другие настольные приложения. Технология DOM использовалась для создания в состояниях динамической модели СОБД, для обработки, привязанных к состояниям модели, данных. Как только приложение меняло текущее состояние на состояние, с которым ассоциированы данные, автоматически создавались динамические DOM-объекты для их обработки. При таком подходе приложение для выполнения естественных операций СУБД по обработке данных занимало оперативную память. Для того, чтобы избежать накладных расходов, связанных с утилизацией памяти, был предложен и разработан другой тип объектов обработки данных, базирующийся на технологии XMLReader, что пригодилось для потоковой обработки данных и разра-

Труды Шестой всероссийской научной конференции "Информационные технологии интеллектуальной поддержки принятия решений", 28-31 мая, Уфа-Ставрополь, Россия, 2018

ботки алгоритмов [6] обработки данных в СОБД. Со временем число источников для СОБД росло, в качестве ответа были предложены новые типы объектов [7], например, JSON для удовлетворения потребностей в обработке данных. Внедрение новых типов объектов было дальнейшим движением в сторону охвата возможностей обработки гетерогенных источников данных, которые были рассмотрены в работах по СОБД [8].

На следующем этапе развития СОБД, когда уже начали развиваться внешние источники гетерогенных данных, представленных в сети в форме независимых от СОБД сервисов, были встроены в динамическую модель и средства удаленного выполнения естественных запросов чтения для обработки данных из удаленных RESTful-сервисов [9, 10].

Поскольку число источников и количество обрабатываемых данных росло, возникла потребность организации виртуальных массивов данных (VDA – Virtual Data Array) [10] структурированных [11] и инвариантно отображаемых на разнородные хранилища данных [12].

В настоящих условиях требуется обработка в виртуальных массивах данных с использованием DPO (Data Processing Objects – объекты обработки данных), таких как DOM- и Smarty-объекты [13–15]. Объем таких данных со временем растет, таким образом возникают источники [16–19] больших данных.

2. Потоковая обработка данных

Собственные источники данных. В основном это данные, получаемые от пользователей, которые доверяют и хранят свои данные веб-приложениям. Источники таких данных могут быть представлены в виде баз данных, хранилищ данных, отдельных файлов и документов в распространенных форматах, а также архивах. Такие данные также генерируются веб-системами для автоматического резервного копирования, образуя таким образом хранилище дампов сведений, уже имеющихся в базах, а также метаданные для них.

Данные веб-сервисов. Есть и другой распространенный вариант работы с источниками, то есть не собственные данные веб-системы, а запрос внешних данных, распространяемых на условиях удаленных сервисов, например, по подписке, так как у Scopus (международная база метаданных и полных текстов научных журналов) или Wikipedia, которая имеет свое собственное хранилище резервных копий – дампов баз данных, туда входят свои собственные сгенерированные данные из содержимого статей и оглавления и других смысловых частей энциклопедии. Дампы занимают значительный размер для каждой своей версии, хранятся в виде архивов, в которые помещены файлы статей, с разветвленной структурой, это могут быть как HTML файлы, так и метаданные в виде XML. Такие данные занимают большой объем даже в архивах. Файлы статей Wikipedia хранимые совместно в архиве занимают большой объем данных, тогда как сами метаданные, представляющие собой один XML-файл, занимает сопоставимый объем данных. Такой файл дампа не может нормально читаться в браузере или текстовом редакторе для просмотра больших файлов, так как возникает нехватка памяти, возникают затруднения в навигации по файлу.

Задача получения и связывания данных из собственных источников и удаленных сервисов. Задача получения таких данных возникает при необходимости обработки связанных сведений собственного источника с данными удаленного сервиса. Примером может выступать деятельность пользователя или его работа в другом сервисе, например, написание статьи Wikipedia или публикация научной статьи в издании, индексируемом Scopus. Естественные потребности увязать такие данные удовлетворяются их получением и обработкой. При получении сведений возникают сложности, связанные с большим объемом данных, передаваемых по сети, при этом во время скачивания расходуется время на их получение.

Обработка полученных данных. При скачивании и локальной распаковке данных время извлечения и обработки увеличивается вдвое и при этом значительная часть внешнего накопителя будет занято. Чтобы обойти такого рода затруднения локальной обработки реализуются так называемые обертки (wrappers) для потокового чтения удаленных данных. Таким образом файлы с данными не скачиваются целиком, а вместо этого открываются на удаленном сервисе и читаются программой. Если данные не велики их можно читать и обрабатывать в памяти (кэшированная обработка) целиком [8], но, если они занимают значительную часть памяти, используется потоковая обработка [8]. Потоковая обработка помогает экономить память и получать результаты порционно в нужном объеме.

Задача получения и обработки данных из сервисов СОБД. Задача обработки больших массивов данных, когда есть файл с большим количеством однообразных объектов в древовидном представлении XML и др. решена и реализована в современных инструментариях информационных систем. В большинстве систем для обработки больших массивов данных реализована и используется технология потоковой обработки данных XMLReader. На данный момент в СОБД нет продвинутых средств способных экономно использовать память для обработки однотипных объектов содержащихся в XML-файлах. Задача получения и обработки больших массивов данных из сервисов, получаемых для СОБД требует решения за счет оснащения динамической модели специализированными средствами потоковой обработки на базе существующих технологий потокового чтения и записи XMLReader и XMLWriter, а также возможности создания DPO-объектов на основе DOM для промежуточного преобразования данных.

Потоки и потоковая обработка. В настоящее время большинство систем программирования реализуют инструментарий для работы с потоками. Поток регламентирует специфику данных, которые будут переданы приложению для обработки, также, как и в обратном направлении в отношении тех данных, которые будут выведены после обработки. Эти потоки ориентируются на файлы, передаваемые по сети, данные, передаваемые в сжатом виде, такие как архивы. В таком случае под потоком подразумевают удаленные ресурсы или их группы, в которых данные хранятся в последовательном виде, как например XML-файл, содержащий последовательно описанные однообразные объекты. Данные организованные последовательно требуют последовательного чтения специализированными средствами, для этого есть частные случаи реализации таких инструментов XMLReader/XMLWriter, ориентированные на работу с XML. В тех условиях, когда есть нестандартные протоколы и типы данных требуется реализовывать дополнительный программный код, называемый оберткой (wrapper) в системе программирования PHP. Есть стандартные обертки, которые имеют свой контекст, где определяется, какие параметры требуются для обработки конкретного известного типа данных. При нехватке оберток для нестандартных протоколов и типов данных системой программирования предоставляется возможность реализовывать свои собственные обертки в виде отдельного скрипта, при этом нет ограничений на количество таких оберток. В данной работе используются стандартные обертки потоков такие как php://output отвечающий за стандартный вывод системы PHP.

3. HSM-модель потоковой обработки

Рассмотрим в общем виде идею HSM-модели для потоковой обработки документов в СОБД. Идея заключается в том, что в модели задаются виртуальные документы, которые, с одной стороны, отображаются на обрабатываемые данные, а с другой, связываются с обработчиком потока данных.

На рис. 1 приведены в общем виде три варианта организации HSM-модели для потоковой обработки данных: с явным заданием цикла потоковой обработки (рис. 1, *a*); с организацией цикла обработки с помощью петлевого перехода (рис. 1, *b*); с неявным заданием цикла обработки (рис. 1, *c*).

В состоянии `sta:Doc-Stream-Processing` заданы два виртуальных документа (VD): `doc:Input` и `doc:Output` – соответственно источник и приемник данных. Во всех трех вариантах декларации виртуальных документов одинаковые, отличия проявляются в организации процесса обработки их данных.

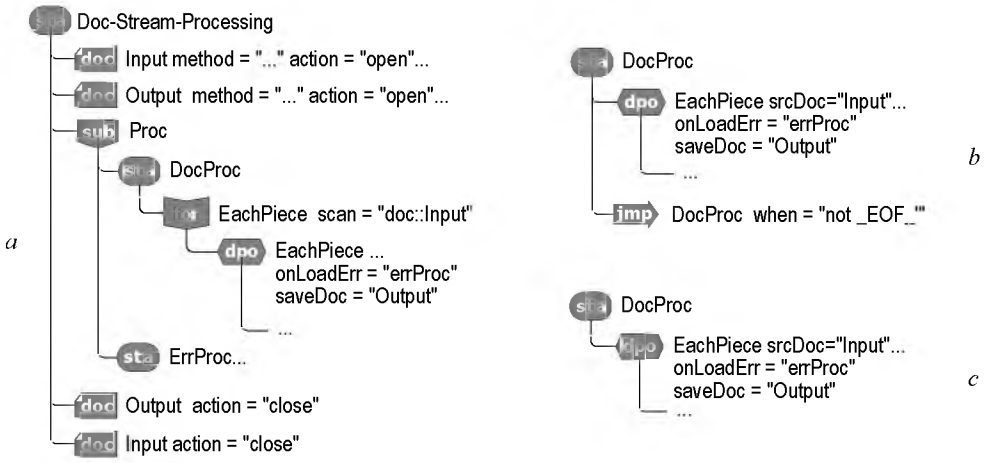

Рис. 1. Общий вид HSM-модели потоковой обработки документов:
a – с явным циклом; *b* – с петлевым переходом; *c* – с неявным циклом

VD задаются с помощью `doc`-элементов – таких же, как и в случае кэшированной обработки. Атрибуты этих элементов (для простоты не показаны на рис. 1) определяют физическое хранилище данных, на которое отображается виртуальный документ. Отличие заключается в задании метода обработки VD с помощью атрибута `method`, который указывает используемый обработчик потока. Атрибут `action` открывает поток с определенными параметрами, задающими те порции, которые будут считываться из источника данных или отправляться в приемник. По завершении обработки открытые потоки закрываются с помощью того же атрибута.

Обработка VD выполняется в субмодели `sub:Proc`, включающей два состояния: `sta:DocProc` – собственно обработка документа; и `sta:ErrProc` – обработка ошибок.

Вариант *a* – с явным циклом потоковой обработки. В этом варианте потоковая обработка организуется с помощью `for`-элемента, предназначенного для циклической интерпретации вложенного в него фрагмен-

та модели (for:EachPiece, см. рис. 1, *a*). Атрибут scan здесь указывает на VD doc:Input, поэтому элемент for задает циклическое чтение порций данных из этого VD и загрузку их по умолчанию в одноименный буфер в виде объекта обработки данных dpo: EachPiece. Этот объект предусматривает переход в состояние sta:ErrProc в случае возникновения ошибки (атрибут onLoadErr) и сохранение содержимого в документе doc:Output в случае успешного завершения обработки (атрибут saveDoc). Собственно обработка содержимого буфера задается вложенными элементами dpo-элемента (на рис. 1 не показаны).

Вариант *b* – с петлевым переходом для организации цикла потоковой обработки. В этом случае dpo-элемент, ссылающийся на VD doc:Input (атрибут srcDoc), задает загрузку в буфер очередной порции данных для обработки. Цикл обработки обеспечивается с помощью элемент перехода jmp:DocProc, который после обработки буфера выполняет повторный переход к состоянию sta:DocProc, пока не будет достигнут конец файла читаемого документа (атрибут when).

Вариант *c* – с неявным заданием цикла потоковой обработки. В этом случае фрагмент модели, относящийся к обработке данных, внешне выглядит точно так же, как и для случая кэшированной обработки документа. При этом подразумевается неявный цикл, такой как в случае варианта *b*, но без петлевого перехода. Таким образом, здесь в ходе интерпретации dpo-элемента происходит обращение к обработчику входного потока, загрузка в буфер порции данных, обработка их и повторение этой процедуры, если не достигнут конец файла.

Сравнение вариантов. В настоящее время сложно сказать, какой из вариантов предпочтительный. В варианте *a* управление используемым обработчиком входного потока (запрос на чтение очередной порции данных) выполняется в ходе интерпретации for-элемента, в вариантах *b* и *c* – в ходе интерпретации dpo-элемента. Вариант *c* расширяет общую концепцию интерпретации dpo-элемента, вводя понятие цикличности. При этом цикличность обработки потока не отражается в модели явно. Однако в этом случае соблюдается принцип инвариантности DPO-обработки по отношению к определению VD [5]: та часть модели, которая задает обработку документа, остается неизменной при изменении отображения на физическое хранилище. Например, модель обработки внешне не изменится, если при определении VD вместо потокового будет задан кэшированный обработчик данных.

4. Пример: потоковая обработка XML

XML-данные являются «родными» для СОБД – именно они использовались в них изначально. Проиллюстрируем потоковую обработку на этом формате данных. На рис. 1 приведен пример HSM-модели потоковой обработки большого XML-документа.

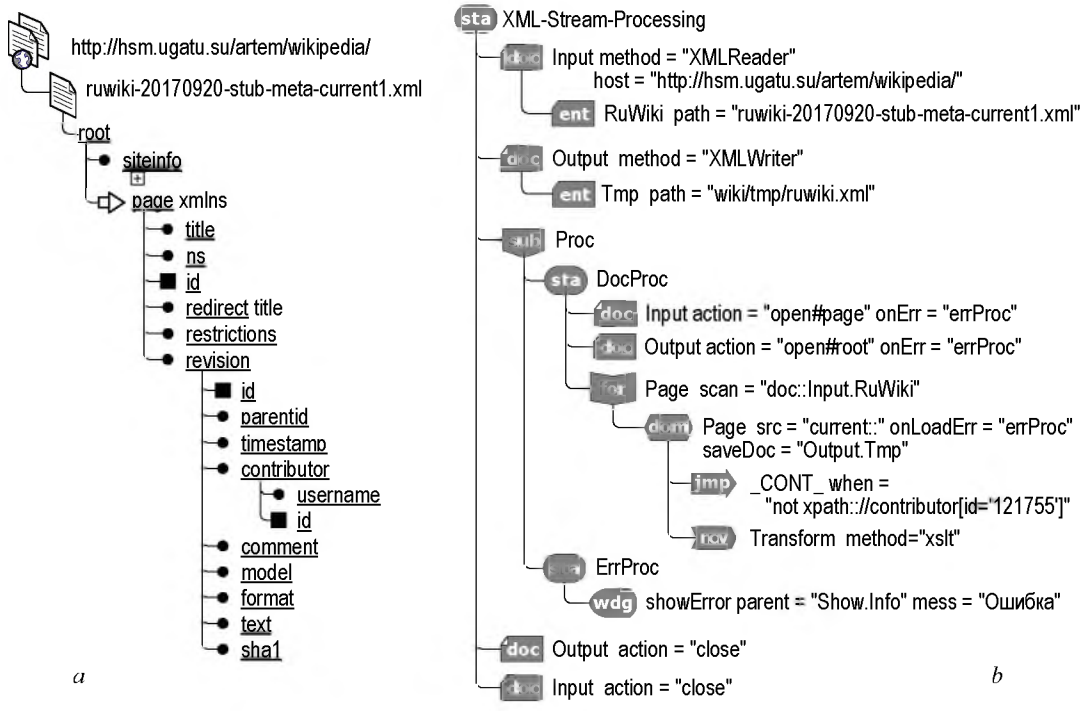

Рис. 2. Пример HSM-модели потоковой обработки XML-данных:
a – структура XML-документа; *b* – модель потоковой обработки

Используемый XML-документ представляет собой файл из архива русскоязычной Википедии. Объем файла превышает 100 Гбайт, его кэшированная обработка на обычном персональном компьютере невоз-

можна. Структура документа приведена на рис. 2, *a*. Корневой элемент `root` содержит дочерний элемент `siteinfo` (содержимое не показано) и множество дочерних элементов `page`, отражающих сведения об изменениях, внесенных в статьи Википедии. Таким образом, данный документ относится к достаточно распространенному виду больших файлов, содержащих большое количество небольших однотипных фрагментов. Это обстоятельство позволяет эффективно организовать потоковую обработку.

Пусть требуется отобрать из документа те элементы `page`, которые принадлежат, скажем, автору с идентификатором 121755, причем нужны не все сведения, а только некоторые, и в другом формате. Таким образом, обработка документа должна включать фильтрацию страниц по содержимому, а также трансформацию содержимого.

На рис. 2, *b* приведена соответствующая HSM-модель. В состоянии `sta:XML-Stream-Processing` заданы два виртуальных мультидокумента: `doc:Input`, `ent:RuWiki`, отображаемый на XML-файл обрабатываемого документа, и `doc:Output`, `ent:Tmp`, отображаемый на результирующий выходной XML-файл. С обоими VD связаны обработчики потока: с первым – XMLReader – средство потокового чтения, а со вторым – XMLWriter – соответственно средство потоковой записи XML-документов.

В субмодели `sub:Proc` выполняется потоковая обработка виртуальных документов по схеме варианта *a* (см. рис. 1, *a*). В состоянии `sta:DocProc` выполняется открытие входного и выходного документов, при этом для входного документа `Input.RuWiki` задается порция чтения – XML-элемент `page`, а для выходного документа `Output.Tmp` – имя корневого XML-элемента – `root`. Элемент `for:Page` циклически читает из входного документа XML-элементы `page` и помещает их по умолчанию в одноименный DOM-объект `dom:Page` для обработки. Атрибут `saveDoc` dom-элемента предписывает по окончании обработки порции данных сохранить результат в выходном документе (отправить в выходной поток).

Собственно обработка порции данных задается элементами `jmp:_CONT_` и `rcv:Transform`. С помощью перехода `jmp:_CONT_` выполняется проверка содержимого обрабатываемого элемента `page`, а именно – значения идентификатора автора (контрибьютора) с помощью XPath-выражения в атрибуте `when`. Если идентификатор автора не совпадает с заданным значением, обработка прерывается, тем самым данный элемент `page` игнорируется в выходном потоке. С помощью приемника `rcv:Transform` выполняется XSL-трансформация содержимого DOM-объекта перед отправкой его в выходной поток. XSLT – это мощное средство преобразования XML-данных на основе таблиц стилей (Stylesheets), как в XML, так и в другие форматы. Метод преобразования задан атрибутом `method`, по умолчанию используется таблица стилей, одноименная с элементом, т. е. `Transform.xsl`. Подробности преобразования здесь опущены для краткости.

При возникновении ошибок происходит переход в состояние `sta:ErrProc`, где формируется соответствующее сообщение пользователю.

5. Заключение

В статье рассмотрены вопросы задания потоковой обработки данных в СОБД на основе декларативных HSM-моделей интеграции гетерогенных источников данных. Для обработки данных в HSM предусматривается, во-первых, задание VD – виртуальных документов, отображаемых на физическое хранилище, во-вторых, задание DPO – объектов обработки данных, в которые загружаются данные из VD.

Предусмотренные ранее возможности HSM ориентированы на кэшированную обработку, при которой VD целиком загружается в оперативную память. Это накладывает ограничения на объем обрабатываемых документов.

Для обработки данных большого объема вычислительные платформы предоставляют возможности потоковой обработки, при которой данные порциями загружаются в память из источника, обрабатываются там и выгружаются в приемник данных. Поставлена задача разработки декларативных средств задания потоковой обработки в рамках HSM.

Введение потоковой обработки потребовало дополнительно предусмотреть в HSM:

• возможность указывать тот или иной потоковый обработчик при задании VD, и специфицировать порции данных, извлекаемых из физического хранилища или помещаемых в него. Для этого в определении виртуального документа предусмотрен атрибут method;

• возможность задания циклической обработки порций данных VD. Для этого предложены три варианта организации: с явным циклом, который обеспечивается `for`-элементом; с циклом, который обеспечивается `jmp`-элементом; с неявным циклом, обеспечиваемым модифицированным `dpo`-элементом.

Возможности задания потоковой обработки продемонстрированы на примере обработки большого XML-документа. Порции документа читаются инструментом XMLReader, загружаются в DOM-объект и обрабатываются в нем, после чего выводятся в выходной поток инструментом XMLWriter.

Таким образом, общая концепция VD/DPO, используемая в СОБД для кэшированной обработки, с небольшими доработками применима и для потоковой обработки больших документов.

Благодарности

Работа поддержана грантом РФФИ № 16-07-00239.

Список использованных источников

1. Гусаренко А. С. Модели создания документов в формате Office Open XML на основе ситуационно- ориентированной базы данных // Прикладная информатика. 2015. № 3 (57) (10). С. 63–76.
2. Миронов В. В., Гусаренко А. С., Диметриев Р. Р., Сарваров М. Р. Создание персонализированных документов на основе ситуационно-ориентированной базы данных // Вестник УГАТУ. 2014. № 4 (65) (18). С. 191–197.
3. Kosar T., Bohra S., Mernik M. Domain-Specific Languages: A Systematic Mapping Study // Information and Software Technology. 2016. Т. 71. С. 77–91.
4. Гусаренко А. С. Усовершенствование модели ситуационно-ориентированной базы данных для взаимодействия с MySQL // Известия высших учебных заведений. Приборостроение. 2016. № 5 (59). С. 355–363.
5. Миронов В. В., Гусаренко А. С., Юсупова Н. И. Отображение виртуальных XML-документов на таблицы MySQL в ситуационно-ориентированных базах данных: «распределенный» подход // Информационные технологии и вычислительные системы. 2017. № 1. С. 77–89.
6. Миронов В. В., Гусаренко А. С., Юсупова Н. И. Ситуационно-ориентированные базы данных: современное состояние и перспективы исследования // Вестник УГАТУ. 2015. № 2 (68) (19). С. 188–199.
7. Гусаренко А. С., Миронов В. В. Smarty-объекты: вариант использования гетерогенных источников в ситуационно-ориентированных базах данных // Вестник УГАТУ. 2014. № 3(64) (18). С. 242–252.
8. Гусаренко А. С., Миронов В. В. Гетерогенные источники документов в ситуационно-ориентированных базах данных // Вестник УГАТУ. 2015. № 4 (19). С. 124–131.
9. Миронов В. В., Гусаренко А. С. Использование RESTful-сервисов в ситуационно-ориентированных базах данных // Вестник УГАТУ. 2015. № 1 (67) (19). С. 232–239.
10. Миронов В. В., Гусаренко А. С., Юсупова Н. И. Ситуационно-ориентированные базы данных: интеграция XML-данных с реляционной СУБД // Системы управления и информационные технологии. 2016. № 3(65). С. 48–56.
11. Миронов В. В., Гусаренко А. С., Юсупова Н. И. Структурирование виртуальных мультидокументов в ситуационно-ориентированных базах данных с помощью entry-элементов // Труды СПИИРАН. 2017. № 53. С. 225–243.
12. Миронов В. В., Гусаренко А. С., Юсупова Н. И. Инвариантность виртуальных данных в ситуационно-ориентированной базе данных при отображении на разнородные хранилища // Вестник компьютерных и информационных технологий. 2017. № 1(151). С. 29–36.
13. Osvaldo S. S. Jr. и др. Developing software systems to Big Data platform based on MapReduce model: An approach based on Model Driven Engineering // Information and Software Technology. 2017. Т. 92. С. 30–48.
14. Cobo M. J., López-Herrera A.G., Herrera-Viedma E. A relational database model for science mapping analysis // Acta Polytechnica Hungarica. 2015. Т. 12. № 6. С. 43–62.
15. Arevalo C. и др. A metamodel to integrate business processes time perspective in BPMN 2.0 // Information and Software Technology. 2016. Т. 77. С. 17–33.
16. Amanatidis T., Chatzigeorgiou A. Studying the evolution of PHP web applications // Information and Software Technology. 2016. Т. 72. С. 48–67.
17. Agh H., Ramsin R. A pattern-based model-driven approach for situational method engineering // Information and Software Technology. 2016. Т. 78. С. 95–120.
18. Mironov V. V., Gusarenko A. S., Yusupova N. I. Situation-oriented databases: document management on the base of embedded dynamic model / CEUR Workshop Proceedings (CEUR-WS.org): Selected Papers of the XI International Scientific-Practical Conference Modern Information Technologies and IT-Education (SITITO 2016), Moscow, Russia, November 25-26, 2016. С. 238–247.
19. Djukic V. и др. Model Execution: An Approach based on extending Domain-Specific Modeling with Action Reports // Computer Science and Information Systems. 2013. Т. 10. № 4. С. 1585–1620.