

UDC 004.032.24

SOLVING THE APPLIED TASKS USING THE HETEROGENEOUS COMPUTING SYSTEMS

T.E. Loshchukhina, V.A. Dorofeev

Tomsk Polytechnic University

The urgency of the discussed issue is caused by the wide spreading of graphics processors and the ability to use them to improve the performance by parallel computing.

The main aim of the study is to identify the positive and negative aspects of using the heterogeneous parallel computing systems for solving non-traditional, i.e. non-graphics tasks involved in application level of information systems; to analyze several hybrid parallel computing technologies and to review the practical example of heterogeneous computing usage.

The methods used in the study: algorithm for finding the optimal route implementation on the selected computing devices, evaluation of the performance of each implementation.

The results: The authors have studied the dependence of the performance of the algorithm for finding optimal route on a number of the original graph nodes and the number of computing devices; they developed the algorithm for updating the working group at the source buffer dimension that is not multiple to the group dimension.

Key words:

Parallel algorithm, hybrid technology, heterogeneous computing, graph model, synchronizing methods.

REFERENCES

1. Daleka V.D., Derevyanko A.S., Kravec O.G., Timanovskaya L.E. *Modeli i struktury dannykh* [Models and data structures]. Har'kov, HGPU, 2000. 241 p.
2. Gergel V.P. *Teoriya i praktika paralelnykh vychisleniy* [Parallel computing theory and practice]. Moscow, Internet-Universitet Informacionnyh Tehnologiy; BINOM. Laboratoriya znaniy, 2007. 423 p.
3. Scarpino M. *OpenCL in action*. S.I., Manning, 2012. 458 p.
4. Mot D. Vvedenie v C++AMP na osnove koda. *Zhurnal MSDN magazine*, 2012. Available at: <http://msdn.microsoft.com/ru-ru/magazine/hh882446.aspx> (accessed 15 April 2013).
5. Mot D. Vvedenie v mozaichnoe zapolnenie C++ AMP. *Zhurnal MSDN magazine*, 2012. Available at: <http://msdn.microsoft.com/ru-ru/magazine/hh882447.aspx> (accessed 15 April 2013).

УДК 004.822; 004.4'2

АНАЛИЗ СТРУКТУРНЫХ ИЗМЕНЕНИЙ В ОНТОЛОГИЯХ НА ЯЗЫКЕ OWL 2

И.А. Заикин

Томский политехнический университет

E-mail: zaikin@tpu.ru

Актуальность работы обусловлена необходимостью коллективной распределённой разработки веб-онтологий, одной из задач которой является сравнение версий онтологий и анализ изменений. Для повышения эффективности совместной работы необходимо предоставить разработчикам обработанную информацию об изменениях в онтологии (какие сущности были изменены, добавлены или удалены в новой версии онтологии), а также о том, какие именно изменения были произведены над каждой сущностью. В работе предложен метод анализа структурных изменений, полученных с использованием прямой семантики языка OWL 2, позволяющий выявлять изменённые, добавленные и удалённые из онтологии сущности, а также сопоставлять каждой затронутой сущности множество изменений. Приведены основные алгоритмы, решающие данную задачу, описание их программной реализации, а также результаты тестирования производительности алгоритмов.

Ключевые слова:

Онтология, OWL 2, аксиома, сущность, изменение.

Введение

Онтологией в информатике называют формальное описание некоторой предметной области или задачи на языке, понятном как людям, так и вычислительным машинам. Одним из самых распространённых языков описания онтологий в настоящее время является язык веб-онтологий OWL 2 [1], стандартизированный консорциумом всемирной паутины (W3C), с использованием которого

ежегодно создаётся множество онтологий, сложность которых также возрастает. Разработка сложных онтологий требует участия не одного, а нескольких специалистов: специалистов по языку онтологий, специалистов в предметной области, специалистов по управлению качеством. Совместная разработка сложных онтологий невозможна без инструментальных средств поддержки коллективной работы. В разработке программного обеспече-

ния одним из таких средств являются системы управления версиями, которые дают возможность вести журнал изменений, отменять нежелательные изменения, сравнивать версии, объединять изменения разных членов команды и облегчают разрешение конфликтов. Одним из важнейших компонентов таких систем является программа для сравнения онтологий, результатом работы которой является набор изменений между двумя версиями. Обычные программы для сравнения текстов опираются на предположение, что порядок строк в тексте имеет значение. В то же время язык OWL 2 не накладывает ограничений на порядок следования аксиом. К тому же одна и та же онтология может быть сохранена в различных форматах. Эти факторы не позволяют использовать для сравнения онтологий существующие программы сравнения текстов.

Язык OWL 2 имеет две семантики: (1) исторически сложившуюся, основанную на RDF [2], где основными понятиями являются ресурсы и отношения между ними, а сложные конструкции представлены с использованием так называемых «пустых узлов», и (2) прямую семантику [3], где основными понятиями являются сущность и аксиома, а для представления сложных конструкций существуют специальные типы аксиом. Сравнение RDF-графов (использование семантики 1) – вычислительно сложный процесс [4], а при наличии большого количества пустых узлов однозначно идентифицировать изменённые триплеты невозможно [5]. В данной работе используется метод структурного сравнения онтологий на основе прямой семантики языка OWL 2. Понятие «структурное сравнение» основано на понятии «структурная эквивалентность», определённом в спецификации OWL 2 [1]. Использование прямой семантики языка OWL 2 даёт возможность не задумываться о пустых узлах, а выполнять сравнение поаксиомно, используя операции над множествами. Однако простые списки добавленных и удалённых аксиом не дают общей картины изменений. Для повышения эффективности совместной работы также необходимо предоставить разработчикам информацию о том, какие сущности были изменены, добавлены или удалены в новой версии онтологии, а также о том, какие именно изменения были произведены над каждой сущностью.

Постановка задачи

Под онтологией в данной работе понимается совокупность $\langle E, A, I, P, d, f \rangle$, где E – конечное неупорядоченное множество сущностей OWL 2 (классов, типов данных, индивидов и свойств), каждая из которых имеет интернационализированный идентификатор (IRI); A – конечное неупорядоченное множество аксиом (некоторых логических утверждений о сущностях); I – конечное неупорядоченное множество ссылок на другие онтологии (импортов); P – конечное неупорядоченное множество пар вида (p_n, p_i) , где p_n – имя префикса – короткая

строка, используемая для сокращения имён сущностей, а p_i – часть IRI, вместо которой может использоваться p_n ; d – идентификатор онтологии, состоящий из IRI самой онтологии и IRI её версии; f – формат онтологии, который может принимать одно из следующих значений: «RDF/XML», «Turtle», «OWL/XML», «OWL Functional Syntax», «OWL Manchester Syntax».

Онтология может содержать только те сущности, на которые ссылается хотя бы одна аксиома. Таким образом, удаление сущности из онтологии подразумевает удаление всех аксиом, ссылающихся на удаляемую сущность, добавление сущности – добавление хотя бы одной аксиомы, ссылающейся на добавляемую сущность, а изменение сущности – добавление не первой аксиомы или удаление не последней аксиомы, ссылающейся на эту сущность.

Пусть исходная версия онтологии v_1 содержит множество сущностей E_1 , а изменённая версия v_2 – множество сущностей E_2 . Задача анализа изменений состоит в решении следующих подзадач:

1. Поиск множества $E' \subseteq E_1 \cup E_2$ сущностей, затронутых изменениями (добавленных, удалённых, изменённых).
2. Разбиение множества E' на три подмножества: E^+ – множество новых сущностей; E^- – множество удалённых сущностей; E^* – множество изменённых сущностей (присутствующих в обеих версиях);
3. Сопоставление каждой изменённой, добавленной и удалённой сущности множества изменений.

Описание метода

Пусть A^U – множество всех возможных аксиом, допустимых с точки зрения синтаксиса языка OWL 2; C^{U+} – множество всех возможных добавлений аксиом, определяемое функцией $\text{AddAxiom}: A^U \rightarrow C^{U+}$, т. е. $C^{U+} = \{\text{AddAxiom}(a) | a \in A^U\}$; C^{U-} – множество всех возможных удалений аксиом, определяемое функцией $\text{RemoveAxiom}: A^U \rightarrow C^{U-}$, т. е. $C^{U-} = \{\text{RemoveAxiom}(a) | a \in A^U\}$; $C^U = C^{U+} \cup C^{U-}$ – множество всех возможных изменений. Определим функцию $\delta: \mathcal{P}(A^U) \times \mathcal{P}(A^U) \rightarrow \mathcal{P}(C^U)$, выполняющую поиск изменений между двумя множествами аксиом $A_1 \subseteq A^U$ и $A_2 \subseteq A^U$ следующим образом:

$$C = \delta(A_1, A_2) = \{\text{RemoveAxiom}(a) | a \in A_1 \setminus A_2\} \cup \{\text{AddAxiom}(a) | a \in A_2 \setminus A_1\},$$

т. е. каждая аксиома, входящая в A_1 , но не входящая в A_2 , считается удалённой, а каждая аксиома, входящая в A_2 , но не входящая в A_1 , считается добавленной.

Определим сигнатуру $\sigma(a)$ аксиомы a как конечное неупорядоченное множество сущностей, на которые ссылается аксиома a . Сигнатура множества аксиом рассчитывается как объединение сигнатур отдельных аксиом:

$$\sigma(\{a_1, a_2, \dots, a_n\}) = \sigma(a_1) \cup \sigma(a_2) \cup \dots \cup \sigma(a_n).$$

Сигнатура изменения равна сигнатуре соответствующей аксиомы:

$\sigma(\text{AddAxiom}(a)) = \sigma(a)$, $\sigma(\text{RemoveAxiom}(a)) = \sigma(a)$.

Сигнатура множества изменений рассчитывается как объединение сигнатур отдельных изменений:

$$\sigma\{c_1, c_2, \dots, c_n\} = \sigma(c_1) \cup \sigma(c_2) \cup \dots \cup \sigma(c_n).$$

Множество сущностей, затронутых изменениями C , можно найти как сигнатуру изменений C : $E' = \sigma(c)$. На рис. 1 множество E' обозначено пунктирным прямоугольником. Исходя из рисунка, можно найти множество удалённых сущностей $E^- = E' \setminus E_2 = E_1 \setminus E_2$, множество добавленных сущностей $E^+ = E' \setminus E_1 = E_2 \setminus E_1$ и множество изменённых сущностей $E^* = E' \cap E_1 \cap E_2$. На рис. 1 также обозначены неизменённые сущности $E^{\parallel} = (E_1 \cap E_2 \setminus E')$.

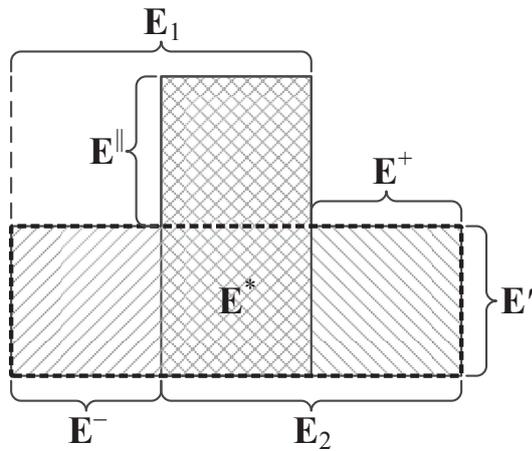


Рис. 1. Соотношение между множествами E_1 (штриховка вправо), E_2 (штриховка влево), E^+ , E^- , E^* , E^{\parallel} и E'

Для каждой сущности e можно найти множество связанных с ней изменений

$$\sigma^{-1}(e) = \{c \in C \mid e \in \sigma(c)\}.$$

Алгоритм извлечения сущностей, затронутых изменениями

Входные данные:

- множество изменений C .

Выходные данные:

- множество затронутых изменениями сущностей $E' = \sigma(c)$.

1. $E' \leftarrow \{\}$
2. Для каждого изменения $c \in C$
3. Для каждой сущности $e \in \sigma(c)$
4. $E' \leftarrow E' \cup \{e\}$

Алгоритм классификации сущностей

Входные данные:

- сигнатура E_1 исходной версии онтологии;
- сигнатура E_2 изменённой версии онтологии;
- множество затронутых изменениями сущностей E' .

Выходные данные:

- множество изменённых сущностей E^* ;
 - множество новых сущностей E^+ ;
 - множество удалённых сущностей E^- .
1. $E^+ \leftarrow \{\}$

2. $E^- \leftarrow \{\}$
3. $E^* \leftarrow \{\}$
4. Для каждой сущности $e \in E'$
5. Если $e \in E_1$
6. Если $e \in E_2$
7. $E^* \leftarrow E^* \cup \{e\}$
8. иначе
9. $E^- \leftarrow E^- \cup \{e\}$
10. иначе
11. $E^+ \leftarrow E^+ \cup \{e\}$

Алгоритм классификации изменений

Входные данные:

- Бинарное отношение $R_{CE} = \{(c, \sigma(c)) \mid c \in C\} \subset C \times \mathcal{P}(E')$, ставящее в соответствие каждому изменению его сигнатуру.

Выходные данные:

- Бинарное отношение $R_{EC} \subset E' \times \mathcal{P}(C)$, ставящее в соответствие каждой сущности множество затрагивающих её изменений.

1. $R_{CE} \leftarrow \{\}$
2. Для каждой пары $(c, E_c) \in R_{CE}$
3. Для каждой сущности $e \in E_c$
4. Если $\exists (e, C_e) \in R_{EC}$
5. $C_e \leftarrow C_e \cup \{c\}$
6. иначе
7. $R_{EC} \leftarrow R_{EC} \cup \{(e, \{c\})\}$

Программная реализация

Предложенные алгоритмы были реализованы на объектно-ориентированном языке программирования Java с использованием библиотеки OWL API [6]. На рис. 2 приведена диаграмма классов программы. Из рисунка видно, что онтология может содержать множество аксиом, а каждая аксиома – множество сущностей, причём одна и та же сущность может встречаться в нескольких аксиомах. Онтология, аксиома и сущность являются подклассами класса OWLObject. Метод getSignature() класса OWLObject реализует функцию σ . Набор изменений ChangeSet состоит из множества изменений OWLAxiomChange, каждое из которых может быть либо добавлением (AddAxiom), либо удалением (RemoveAxiom) аксиомы. Конструктор класса ChangeSet принимает два экземпляра OWLOntology и выполняет их поаксиомное сравнение (реализует функцию δ). Результат сравнения можно получить, вызвав метод getAxiomChanges(). Класс EntityExtractor реализует алгоритм извлечения сущностей, затронутых изменениями. Метод getEntities() предоставляет доступ к множеству извлечённых сущностей. Класс EntityClassifier реализует алгоритм классификации сущностей и содержит методы getNewEntities(), getRemovedEntities() и getModifiedEntities() для доступа к новым, удалённым и изменённым сущностям соответственно, а также метод getAllEntities() для удобного доступа ко всем затронутым изменениями сущностям. Класс ChangeClassifier реализует алгоритм классификации изменений и предоставляет метод getChangesByEntity() для получения

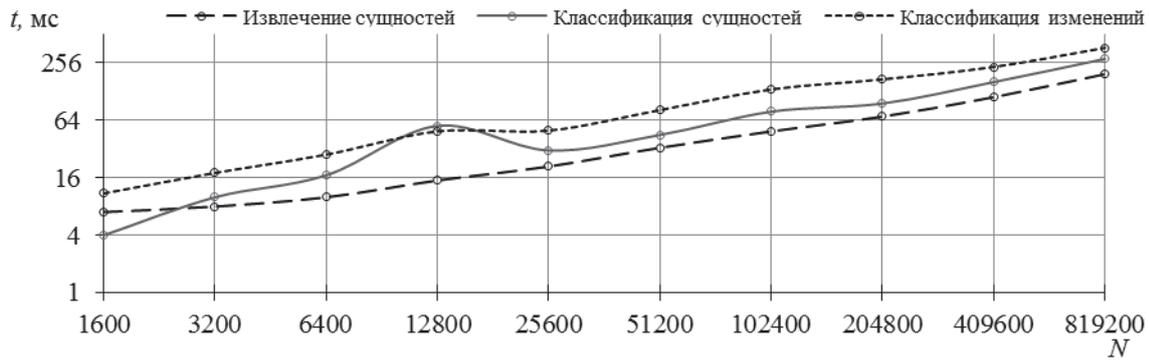


Рис. 3. График зависимости времени работы алгоритмов от размера онтологий

других форматов. Для таких онтологий не требуется анализ при каждом изменении, однако приведённые алгоритмы могут быть применены и к ним.

Заключение

Предложенный метод анализа изменений позволяет выделять сущности, фигурирующие в изменениях, классифицировать их на новые, удалённые и изменённые, а также находить изменения, соответствующие этим сущностям. Результаты решений по предложенному методу в дальнейшем предполагается использовать при построении системы управления проектами, предназначенной для разработ-

ки онтологий. Данный метод даст возможность отслеживать, когда та или иная сущность была введена, удалена или изменена, что позволит автоматизировать описание изменений разработчиками онтологий, а также позволит им быстро находить требуемые изменения в журнале изменений.

Разработанное программное обеспечение использовалось при построении онтологии предметной области «Интеллектуальное месторождение».

Разработка проводилась при финансовой поддержке Министерства образования и науки Российской Федерации, в рамках финансирования работ по Государственному контракту 14.515.11.0047.

СПИСОК ЛИТЕРАТУРЫ

1. Motik B., Patel P.F., Parsia B. OWL 2 Web Ontology Language structural specification and functional style syntax // World Wide Web Consortium. 2009. URL: <http://www.w3.org/TR/owl2-syntax/> (дата обращения: 31.05.2013).
2. Schneider M. OWL 2 Web Ontology Language RDF-Based Semantics // World Wide Web Consortium. 2009. URL: <http://www.w3.org/TR/owl2-rdf-based-semantics/> (дата обращения: 11.11.2012).
3. Motik B., Patel-Schneider P.F., Cuenca Grau B. OWL 2 Web Ontology Language Direct Semantics // World Wide Web Consortium. 2009. URL: <http://www.w3.org/TR/owl2-direct-semantics/> (дата обращения: 11.11.2012).
4. Carroll J. Matching RDF Graphs. 26.11.2001. URL: <http://www.hpl.hp.com/techreports/2001/HPL-2001-293.pdf> (дата обращения: 23.10.2012).
5. DeltaView // ESW Wiki. 12.11.2007. URL: <http://esw.w3.org/DeltaView> (дата обращения: 31.05.2013).
6. Horridge M., Bechhofer S. The OWL API: A Java API for Working with OWL 2 Ontologies // CEUR Workshop Proceedings (OWLED 2009: 6th OWL Experiences and Directions Workshop, Chantilly, Virginia, October 2009). – 2009. – V. 529. – P. 53–62.

Поступила 05.06.2013 г.

ANALYSIS OF STRUCTURAL CHANGES IN OWL 2 ONTOLOGIES

I.A. Zaikin

Tomsk Polytechnic University

The urgency of the discussed problem is caused by the need to develop web ontologies in a distributed collaborative manner. One of the tasks is the ontology version comparison and change analysis. The developers require processed information on ontology changes in order to collaborate more efficiently (what entities were changes, added or removed from the ontology), as well as the information on changes performed on each specific entity. This paper proposes a way of analyzing structural changes obtained when using OWL 2 direct semantics, which allows us to reveal entities that were modified, added or removed from the ontology, as well as to find changes related to each modified entity. The author shows the main algorithms, which solve this task, software implementation of the algorithms, and performance testing results.

Key words:

Ontology, OWL 2, axiom, entity, change.

REFERENCES

1. Motik B., Patel P.F., Parsia B. *OWL 2 Web Ontology Language structural specification and functional style syntax*. World Wide Web Consortium. 2009. Available at: <http://www.w3.org/TR/owl2-syntax/> (accessed 31 May 2013).
2. Schneider M. *OWL 2 Web Ontology Language RDF-Based Semantics*. World Wide Web Consortium. 2009. Available at: <http://www.w3.org/TR/owl2-rdf-based-semantics/> (accessed 11 November 2012).
3. Motik B., Patel-Schneider P.F., Cuenca Grau B. *OWL 2 Web Ontology Language Direct Semantics*. World Wide Web Consortium. 2009. Available at: <http://www.w3.org/TR/owl2-direct-semantics/> (accessed 11 November 2012).
4. Carroll J. *Matching RDF Graphs*. 26.11.2001. Available at: <http://www.hpl.hp.com/techreports/2001/HPL-2001-293.pdf> (accessed 23 October 2012).
5. *DeltaView*. ESW Wiki. 12.11.2007. Available at: <http://esw.w3.org/DeltaView> (accessed 31 May 2013).
6. Horridge M., Bechhofer S. The OWL API: A Java API for Working with OWL 2 Ontologies. *CEUR Workshop Proceedings (OWLED 2009: 6th OWL Experiences and Directions Workshop*. Chantilly, Virginia, October 2009, vol. 529, pp. 53–62.