Krassimir Markov, Vitalii Velychko, Oleksy Voloshin
(editors)

Information Models of Knowledge

I T H E A®
K I E V – S O F I A
2010

Krassimir Markov, Vitalii Velychko, Oleksy Voloshin (ed.)

Information Models of Knowledge

ITHEA®

Kiev, Ukraine – Sofia, Bulgaria, 2010

ISBN 978-954-16-0048-1

First edition

Recommended for publication by The Scientific Concil of the Institute of Information Theories and Applications FOI ITHEA

ITHEA IBS ISC: 19.

This book maintains articles on actual problems of research and application of information technologies, especially the new approaches, models, algorithms and methods fot information modeling of knowledge in: Intelligence metasynthesis and knowledge processing in intelligent systems; Formalisms and methods of knowledge representation; Connectionism and neural nets; System analysis and sintesis; Modelling of the complex artificial systems; Image Processing and Computer Vision; Computer virtual reality; Virtual laboratories for computer-aided design; Decision support systems; Information models of knowledge of and for education; Open social info-educational platforms; Web-based educational information systems; Semantic Web Technologies; Mathematical foundations for information modeling of knowledge; Discrete mathematics; Mathematical methods for research of complex systems.

It is represented that book articles will be interesting for experts in the field of information technologies as well as for practical users.

General Sponsor: Consortium FOI Bulgaria (www.foibg.com).

Printed in Ukraine

ISBN 978-954-16-0048-1

C\o Jusautor, Sofia, 2010

К ИНТЕГРАЦИИ ОНТОЛОГИЙ ПРЕДМЕТНЫХ ОБЛАСТЕЙ

Александр Палагин, Андрей Михайлюк, Виталий Величко, Николай Петренко

***Аннотация**: Рассматривается проблема системной интеграции онтологий предметных областей с точки зрения формирования общего интегрированного пространства трансдисциплинарных знаний. Разработана методика и алгоритм указанной интеграции.*

***Ключевые слова**: онтология предметной области, системная интеграция.*

***ACM Classification Keywords**: I.2 ARTIFICIAL INTELLIGENCE - I.2.4 Knowledge Representation Formalisms and Methods.*

Введение

Новым шагом развития междисциплинарных научных исследований, и теории баз знаний в частности, должно стать теоретически обоснованное объединение (или системная интеграция) уже разработанных как общедоступных онтологий, так и коммерческих баз знаний для разнообразных прикладных задач, проблем, целых предметных областей и трансдисциплинарных знаний общего характера. В [Палагін 2009] сущность системной интеграции сформулирована следующим образом: «Устойчивые знания совокупности научных дисциплин можно представить в форме интегрированной иерархической сети научных теорий (разного уровня развития, содержательности и охвата действительности), составляющие которых, возможно, связаны посредством общих объектов действительности». Там же говорится о цели междисциплинарных исследований – приближение к построению общенаучной картины мира – а также о системной интеграции знаний (онтологий) как одной из важных задач в достижении указанной цели.

Системная интеграция двух и более онтологий представляется далеко не тривиальной задачей. Её алгоритмы не проработаны, требуют тщательной проверки и апробации на представительном множестве онтологий.

Формально онтология представляется тройкой $\langle X, R, F \rangle$, где X – множество концептов, R – множество отношений между концептами, F – функции интерпретации концептов из множества X и отношений из R. Даная модель носит общий характер, в то время, как на практике пользуются более точными моделями. Одной из таковых является модель Protégé Frames [Noy 2000 (1)], разработанная для использования в редакторе онтологий Protégé [Sachs 2006] и базирующаяся на фреймовом представлении и протоколе OKBC (Open Knowledge Base Connectivity) [Chaudhri 1998]. Известны и другие применяемые на практике модели, активно используемые в онтологическом инжиниринге, особое место среди которых занимает язык описания онтологических знаний OWL [Dean 2004], разработанный для онтологизации веб-пространства и принятый в качестве стандарта консорциумом W3C. Однако, все современные практически-ориентированные модели эквивалентны модели $\langle X, R, F \rangle$ или являются ее частными (ограниченными) представлениями, а, следовательно, могут быть представлены в ней, например, согласно схеме соотнесения общепринятой модели и фреймовой модели Protégé (рис. 1). Исходя из этого, для описания операций над онтологиями можно использовать любую из моделей.

В данной работе будет использована фреймовая модель Protégé, так как она имеет высокую степень абстрактности, достаточно распространена и описана во множестве публикаций и документации разработчиков [Protégé project и др.], имеет формальные средства описания в популярном языке OWL [Knublauch 2004], и, кроме того, редактор, основанный на этой модели, является общедоступным как в исходных кодах, так и в бинарном виде для большинства современных платформ [Download Protégé], что дает возможность применять его на практике.

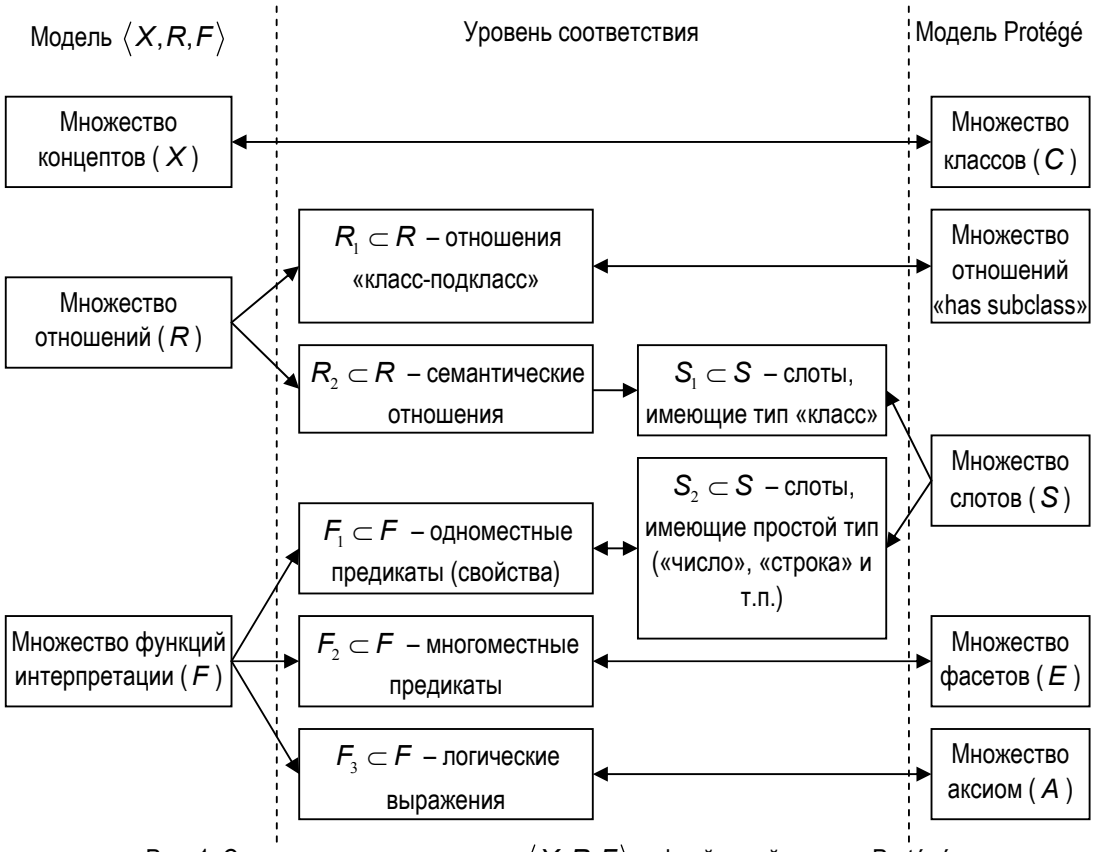

Рис. 1. Схема соотнесения модели $\langle X, R, F \rangle$ и фреймовой модели Protégé.

Постановка задачи

Каждая предметная область (ПдО) может содержать подмножество классов, эквивалентных классам понятий других ПдО. Это могут быть как классы, обозначающие общенаучные понятия, так и классы, обозначающие понятия, специфические для каждой ПдО. На основе отождествления некоторых классов (понятий) разных ПдО возможно построение их общей онтологии. На сегодняшний день известно множество онторедакторов, большинство из которых реализуют процедуру построения общей онтологии из двух исходных [Овдей 2004]. Практическое применение такого объединения очевидно – от добавления в онтологию нового, ранее не разработанного фрагмента ПдО, полученного от других разработчиков, до построения прототипа общего интегрированного пространства трансдисциплинарных знаний.

Ряд задач все еще остается открытым, а их решение, как правило, предоставляется пользователю системы (инженеру по знаниям). Кроме того, большинство решений, принимаемых указанными системами относительно эквивалентности классов, являются очевидными, тогда как более сложные случаи подобия пользователь вынужден определять вручную. Целью данной работы является расширение процедур обработки онтологий, качественное улучшение известных программных продуктов манипулирования онтологиями путем повышения уровня автоматизации их работы и адекватности предлагаемых пользователю действий. Для этого разрабатывается, обосновывается и описывается алгоритм построения результирующей онтологии из двух исходных на основе результатов сравнения классов. Кроме того, предлагается обобщение задачи объединения онтологий до процедуры интеграции онтологий, описывающих различные ПдО одного и того же домена предметных областей.

Общий подход к объединению онтологий

Из известных операций над онтологиями [Овдей 2004] объединение онтологий является одной из наиболее важных и часто используемых пользователями процедурой. Суть указанной процедуры состоит в построении результирующей онтологии из двух исходных с сохранением первичных онтологических знаний в таком виде, чтобы результирующая онтология была максимально связанной (содержала все отношения между классами) и не содержала дублирующих данных (например, эквивалентных классов).

В наилучшем варианте, алгоритм объединения онтологий ориентирован на автоматическое выполнение необходимых действий без какого-либо участия пользователя. По окончании его работы результат предъявляется пользователю системы в любой системе визуализации (возможно, не связанной с системой объединения). С другой стороны, наиболее популярные на сегодня приложения объединения онтологий [Овдей 2004] предлагают полуавтоматический режим. Он заключается в интерактивном объединении пользователем пар классов, принадлежащих объединяемым онтологиям, объединяя таким образом сами онтологии «поэлементно». Основная роль системы при этом заключается в автоматическом выполнении операций, связанных с объединением классов (создание новых «объединяющих» классов, копирование их слотов, установление родовидовой зависимости, переименование классов и слотов и т.д.). Кроме того, в некоторых подсистемах (плагинах или модулях онторедакторов) объединения онтологий [Noy 2003, McGuinness 2000] помимо выполнения операций, связанных с объединением онтологий, пользователю предлагается набор возможных на данном этапе операций («объединить классы», «скопировать слот» и т.п.).

Алгоритм объединения онтологий можно разделить на три относительно независимых этапа, на каждом из которых учитывается полученный не предыдущем этапе результат:

1. Установление степени эквивалентности между классами двух онтологий. На данном этапе производится сравнение всех классов одной онтологии со всеми классами другой. Качество процедуры сравнения существенно влияет на качество объединения онтологий.

2. Поэлементное объединение множеств классов и множеств слотов исходных онтологий. Основная часть, по-сути, и являющаяся объединением, основывается на степени эквивалентности, вычисленной на предыдущем этапе для каждой пары классов. Этот этап, как правило, выполняется полностью автоматически, так как не требует принятия решений относительно семантики классов и полностью формализован. Выбор алгоритма работы данного этапа объединения онтологий влияет на степень интеллектуальности всей процедуры. Под интеллектуальностью процедуры объединения онтологий понимается возможная степень сложности результирующих графов – родовидовых иерархий (классы, соединенные отношениями «класс-подкласс») и семантических сетей (классы, соединенные семантическими отношениями), которые могут быть построены при объединении, возможную степень их отличия от соответствующих графов исходных онтологий.

3. Проверка результата объединения на корректность. Под корректностью понимается непротиворечивость результирующей онтологии и степень ее соответствия ожиданиям пользователя. Последнее может быть полностью реализовано только самим пользователем, тогда как критерий противоречивости может быть определен формально. Возможные типы противоречий определяются моделью описания онтологии. В простом варианте – это противоречия, связанные с уникальностью имен классов и слотов [Noy 2000 (2)]. В общем случае, противоречия определяются ложностью аксиом и функций интерпретации. Поскольку большинство онторедакторов имеют ограниченные возможности задания логических зависимостей между классами и их отношениями, противоречия таких типов не проверяются в связи с отсутствием в модели функций интерпретации в явном виде (модель Protégé, например, сводит задание функций интерпретации к заданию фасетов).

Сравнение классов

Для реализации первого этапа процедуры объединения онтологий необходимо разработать операцию сравнения классов двух онтологий. Входными параметрами для операции выступают исходные онтологии (O и O') и пара сравниваемых классов ($C \in O$ и $C' \in O'$). Результатом указанной операции является логическое значение («истина» или «ложь»), соответствующее ответу на вопрос «Эквивалентен ли класс C классу C'?». Исходя из сложности внутренней структуры класса и его отношений с другими классами, отметим нетривиальность задачи сравнения классов. Прежде всего, следует определить, на основании чего можно утверждать, что некоторый класс эквивалентен или подобен другому классу. Поэтому, первой частью разработки операции сравнения классов является выбор критериев сравнения.

Наиболее простым способом сравнения классов является *сравнение по имени*. Данный подход является основным критерием большинства существующих модулей объединения онтологий [Овдей 2004]. В наиболее развитых из них (таких как PROMPT) существует возможность нечеткого сравнения имен классов и сравнения с учетом синонимов [Ehrig 2005], однако в случае сложных синонимичных и омонимичных конструкций сравнение классов по данному критерию дает неверные результаты. Причиной таких ошибок является предположение о роли класса в онтологии по его имени, тогда как имя не определяет класс (как термин не определяет понятия), а служит удобной для человека меткой класса.

Более сложным вариантом сравнения классов является сравнение по содержанию (составу), однако во многих случаях отождествление классов (понятий) на основе поэлементного равенства всех их экземпляров является ошибочным, описание причин и примеры чего представлены в [Войшвилло 2007]. Несмотря на это, сравнение объемов классов активно используется в современных системах как вспомогательная информация для установления подобия между ними [Noy 2000 (2), Noy 2001].

Согласно [Войшвилло 2007, Ивлев 2001], понятие полностью определяется своим *содержанием*, что следует из самого определения содержания понятия, а поскольку класс в модели Protégé аналогичен понятию, то это же должно быть справедливо и для классов. Исходя из этого, можно заключить, что при сравнении классов наиболее важной составляющей в их структуре является его содержание, т.е. множество слотов, доменом которых является данный класс. Однако, отношения между множествами (в данном случае, множествами слотов) не ограничиваются отношениями эквивалентности и различия – последнее может представлять включение одного множества в другое, включение второго множества в первое, непустое пересечение двух множеств и отсутствие общих элементов (полное различие). Относительно множеств слотов данных двух классов указанные отношения можно трактовать следующим образом:

1. Совпадение всех слотов класса C_1 со всеми слотами класса C_2 (поэлементное равенство множеств $\{S\}_1$ и $\{S\}_2$) означает равенство содержания двух классов. Это в свою очередь, свидетельствует об *эквивалентности* самих классов. Назовем данный результат «эквивалентность».

2. Включение множества $\{S\}_1$ слотов класса C_1 во множество $\{S\}_2$ слотов класса C_2 ($\{S\}_1 \subset \{S\}_2$) свидетельствует об обобщении содержания класса C_2 до содержания класса C_1. Это означает, что сам класс C_1 является более общим по отношению к классу C_2. Исходя из принципа обратной зависимости содержания и объёма понятий (увеличение содержания влечет уменьшение объёма и наоборот [Войшвилло 2007, Ивлев 2001]), можно заключить, что класс C_1 является одним из *надклассов* класса C_2. Назовем данный результат «обобщение».

3. Включение множества $\{S\}_2$ слотов класса C_2 во множество $\{S\}_1$ слотов класса C_1 ($\{S\}_1 \supset \{S\}_2$) является случаем, обратным к случаю (2), и означает, что класс C_1 является *подклассом* C_2, т.е. понятие, представляемое классом C_1, является уточнением понятия, представляемого классом C_2. Назовем данный результат «уточнение».

4. Пересечение множеств слотов классов C_1 и C_2 ($\{S\}_1 \cap \{S\}_2 \neq \varnothing$) свидетельствует о наличии общих слотов. Это означает, что существует некоторый класс C, являющийся надклассом для классов C_1 и C_2, а сами классы принадлежат одному уровню иерархии. Назовем данный результат «частичная эквивалентность».

5. Пустое пересечение множеств слотов классов C_1 и C_2 ($\{S\}_1 \cap \{S\}_2 = \varnothing$) свидетельствует о том, что у первого общего для C_1 и C_2 надкласса отсутствуют слоты. Таким надклассом может быть корень родовидового дерева («Универсум» или класс «:THING» в Protégé) или классы категориального уровня (такие как «абстрактное», «материальное», «процесс», «сущность» и т.д. [Палагін 2006]). В этом случае, такие классы можно считать полностью различными. Назовем данный результат «различие».

Описанный механизм сравнения определяет взаимное родовидовое отношение двух классов, что является обобщением отношения эквивалентности и вместо двух вариантов результатов сравнения классов предполагает пять. Однако, такой механизм не применим к классам с пустым множеством слотов – в этом случае (согласно описанному механизму) такие классы должны быть эквивалентны, что, однако, в большинстве случаев неверно. Пустое множество слотов класса говорит о том, что понятие, представляемое классом, не требует описания его содержания в явном виде, либо что его содержание неизвестно. Последний случай следует отнести к некорректности входных данных, так как алгоритм ориентирован на объединение полных онтологий. Случай же с необязательностью указания содержания некоторого понятия возникает при уверенности инженера по знаниям, составляющего онтологию, что оно (понятие) известно интерпретатору онтологии (системе обработки онтологий или другому инженеру по знаниям). В таком случае понятие, а следовательно и представляющий его класс, должны однозначно идентифицироваться своим именем, поэтому предлагается *классы без слотов сравнивать по именам*.

Объединение и интеграция онтологий

Основываясь на результатах поэлементного сравнения классов, система может перейти к непосредственному объединению онтологий. Базовый механизм объединения, используемый в современных системах обработки онтологии [Овдей 2004], заключается в поиске классов, эквивалентных, по мнению системы или решению пользователя. Эквивалентные классы добавляются в результирующую онтологию, а их содержание представляет собой множественно-логическое объединение слотов и фасетов исходных классов. При необходимости слоты могут переименовываться, фасеты проверяться на совместимость, и т.д. Именно часть корректного переноса содержания исходных классов в результирующий считается основной задачей процедуры объединения. Несомненно, наличие такой процедуры значительно упрощает процесс получения новой онтологии из двух исходных, по-сравнению с «ручным» режимом. Однако, такой подход носит во многом механический характер «склеивания» отдельных классов без попытки интеллектуализировать этот процесс.

В данной работе предлагается расширить возможности процедуры объединения онтологий за счет обобщения операций, предлагаемых пользователю в существующих системах. Этап перенесения слотов и фасетов исходных классов в соответствующие классы результирующей онтологии остается неизменной, а в большинстве случаев даже упрощается, поэтому детально не будет описан в данной работе (см. [Noy 2000 (2)]). Итак, как было отмечено выше, обобщением отношения эквивалентности классов является отношение родовидовой зависимости: вместо взаимного позиционирования двух классов как эквивалентных или различных предлагается рассматривать эти классы как пару эквивалентных, пару частное-общее, общее-частное, пару классов с общей частью или же пару полностью различных классов (без общей части). Очевидно, пять вариантов взаимного отношения классов как результата операции их сравнения приводят к пяти различным операциям над классами на этапе объединения онтологий:

– если классы эквивалентны, то они представляют одно и то же понятие в онтологии, следовательно, должны быть «склеены» в один;

– если класс одной онтологии является обобщением соответствующего класса другой онтологии, такие классы должны представлятся как класс и подкласс соответственно, причем совпадающие слоты должны быть удалены из подкласса, так как они будут унаследованы от надкласса (т.к. отношение "класс-подкласс" является отношением частичного порядка);

– если класс одной онтологии является уточнением соответствующего класса другой онтологии, такие классы должны представлятся как подкласс и класс соответственно, причем совпадающие слоты должны быть удалены из подкласса, так как они будут унаследованы от надкласса;

– если классы двух онтологий частично эквивалентны, то они представляют собой схожие понятия, то есть должны иметь общий надкласс, являющийся их обобщением (заметим, этот надкласс не присутствовал ни в одной из исходных онтологий), при этом совпадающие слоты должны быть удалены из подкласса, так как они будут унаследованы от обобщающего класса;

– если классы различны, они должны быть скопированы в результирующую онтологию.

Как было отмечено выше, при объединении применяется итеративный подход, когда после выполнения операции над текущей парой классов система проводит повторное сравнение необработанных классов, исходя из результатов выполненной операции, и предлагает (или сразу выполняет) наиболее подходящую на этом этапе операцию из описанных выше. Поскольку классы исходных онтологий либо полностью копируются в результирующую, либо переносятся в нее с изменением родовидовых связей и удалением общих слотов, то и сравнение классов следует выполнять в рамках результирующей онтологии. В таком случае, возможно построение новой онтологии на базе одной из исходных путем введения в нее классов из второй исходной онтологии.

Описанная процедура качественно отличается от существующих процедур *объединения* – вместо объединения онтологий «склеиванием» эквивалентных вершин происходит взаимное «проникновение» онтологий со значительным изменением исходной структуры, но полным сохранением исходных онтологических знаний. Поэтому данную процедуру более точно было бы называть *интеграцией* (а не объединением) онтологий. Основой процедуры интеграции онтологии в таком случае является не «склеивание» эквивалентных вершин, а интеграция каждого класса интегрируемой онтологии в базовую.

Алгоритм процедуры интеграции онтологий. Алгоритм процедуры интеграции онтологий удобно разбить на три части: общая часть, интеграция родовидовых иерархий (интеграция с учетов отношений класс-подкласс) и интеграция классов (интеграция с учетом слотов как элементов содержания классов). Представим блок-схемы этих трех частей процедуры интеграции онтологий (согласно [ГОСТ 19.701-90] и [ISO 5807:1985]) и опишем их алгоритмы.

Общая часть процедуры выполняет подготовительную и завершающую части интеграции. Кроме того, она вызывает процедуру поэлементной нисходящей интеграции (интеграция иерархий). Поэтому назовем эту часть процедуры общим названием «Интеграция онтологий» и представим блок-схему ее алгоритма на рис. 2 (а).

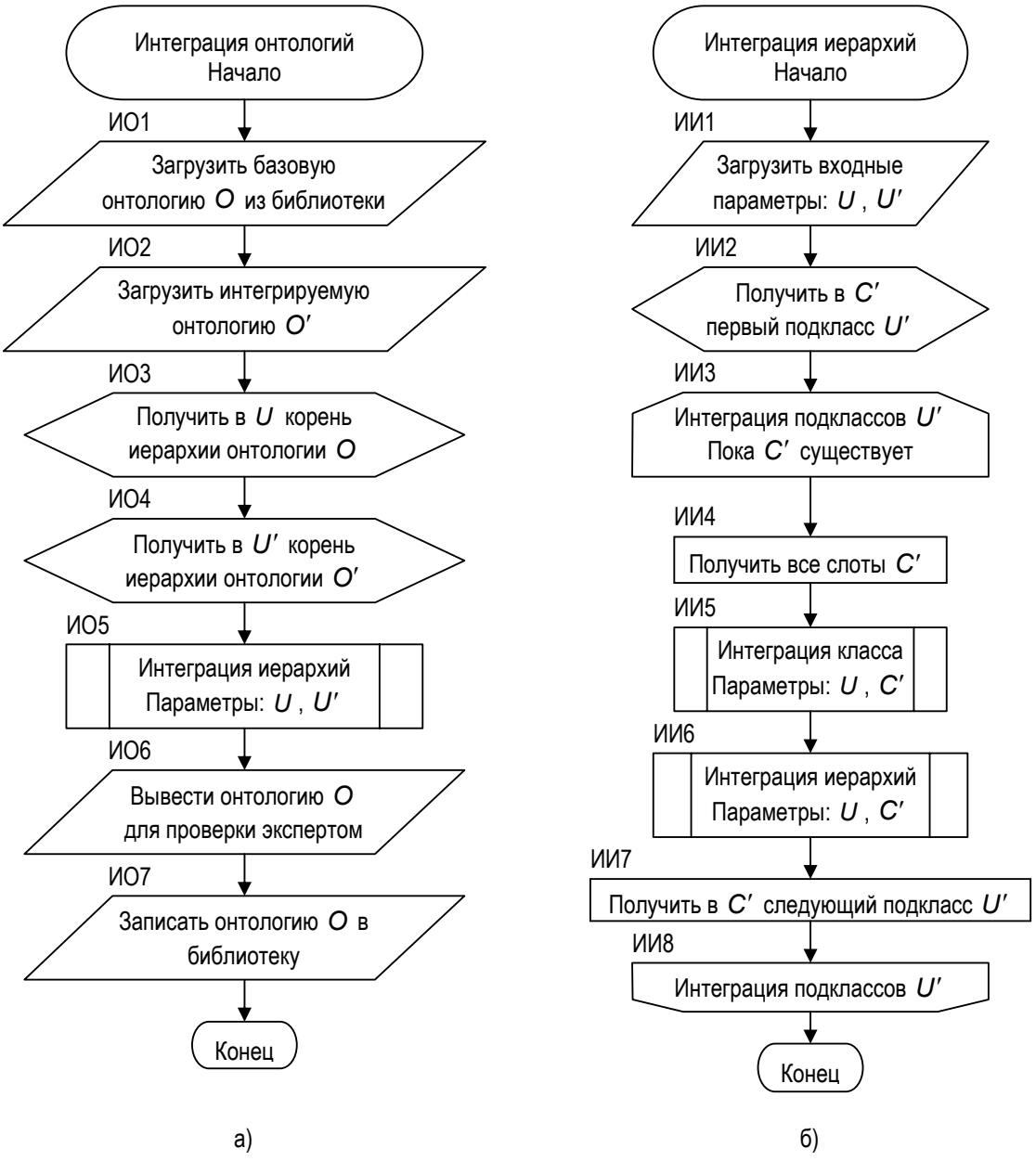

Рис 2. Блок-схема процедур: (а) «Интеграция онтологий», (б) «Интеграция иерархий»

ИО1. Выполняется загрузка базовой онтологии O из библиотеки онтологий ПдО в модуль системной интеграции онтологий.

ИО2. Выполняется загрузка онтологии O'. Источником может быть web-страница с RDF разметкой, носитель данных с сохраненной ранее онтологией, другая система обработки онтологий (например, система поиска документов и составления их онтологий).

ИО3. Выполняется инициализация корневой вершины U базовой онтологии O.

ИО4. Выполняется инициализация корневой вершины U' интегрируемой онтологии O'.

ИО5. Вызов и передача управления подпрограмме «Интеграция иерархий» для корневых классов базовой и интегрируемой онтологии.

ИО6. Процедура «Интеграция онтологий» передает результат работы подпрограммы «Интеграция иерархий» в подсистему визуализации (например, Protégé) для просмотра и проверки инженером по знаниям правильности выполненной интеграции.

ИО7. Выполняется запись результирующей онтологии в соответствующую библиотеку.

Рассмотрим алгоритм интеграции иерархий, представленный на рис.2б.

Интеграция структур классов исходных онтологий представляется как интеграция родовидовой иерархии, начиная с корня иерархического дерева. Эту задачу, в свою очередь, удобно выполнять рекурсивно: интегрировать корень иерархии (входящий класс) в базовую иерархию (также заданную своим корнем – входным параметром) и поддеревья каждого из подклассов данного класса вызовом этой же процедуры с текущим подклассом данного класса в качестве вершины иерархии. Для возможности рекурсивного обращения данная часть алгоритма системной интеграции выделена в отдельную процедуру.

ИИ1. Процедура «Интеграция иерархий» загружает входные параметры – классы U, U'. Класс U задает вершину родовидового дерева базовой онтологии, в которую будет интегрироваться иерархия, заданная корневым классом U'. Сам класс U' не интегрируется в иерархию U, так как представляет собой прототип класса U в интегрируемой онтологии.

ИИ2. Задание временной переменной C' (указателя на класс – итератора для прохождения по списку непосредственных подклассов U') начального значения – первого непосредственного подкласса класса U' в списке подклассов класса U'.

ИИ3. Начало цикла с предусловием. В цикле осуществляется интеграция текущего подкласса C' класса U' в иерархию, заданную классом U, и всей иерархии, заданной подклассом C'. Цикл выполняется, пока не будет достигнут конец списка подклассов класса U', то есть подкласс C' как элемент списка «удачно» извлечен (предполагается, что при достижении конца списка операция извлечения следующего его элемента вернет несуществующий «нулевой» элемент).

ИИ4. В список слотов класса C' добавляются слоты всех его надклассов, реализуя, таким образом, принцип наследования. Данный блок необходим, так как подпрограмма «Интеграция класса» интегрирует класс с полным списком слотов в заданную иерархию.

ИИ5. Класс C' интегрируется в иерархию базовой онтологии с вершиной U вызовом процедуры «Интеграция класса».

ИИ6. Рекурсивный вызов процедуры «Интеграция иерархий» относительно класса U и текущего подкласса C' для интеграции поддерева интегрируемой онтологии, заданного вершиной C', в иерархию базовой онтологии, заданной классом U.

ИИ7. Итерирование по списку подклассов класса U' – получение следующего элемента списка подклассов класса U' и запись его в переменную C'.

ИИ8. Переход на начало цикла (блок **ИИ3**) «Интеграция подклассов» с текущим значением C'.

Интеграция классов исходных онтологий заключается в размещении данного класса интегрируемой онтологии в иерархию базовой онтологии. При этом учитываются варианты взаимного отношения классов, описанные в подпункте «Базовый алгоритм сравнения классов», и применяются операции, описанные в начале данного пункта. Общая схема работы алгоритма состоит во вставке интегрируемого класса в базовую иерархию на максимально низкий уровень. Проход вниз по иерархии реализован рекурсивно.

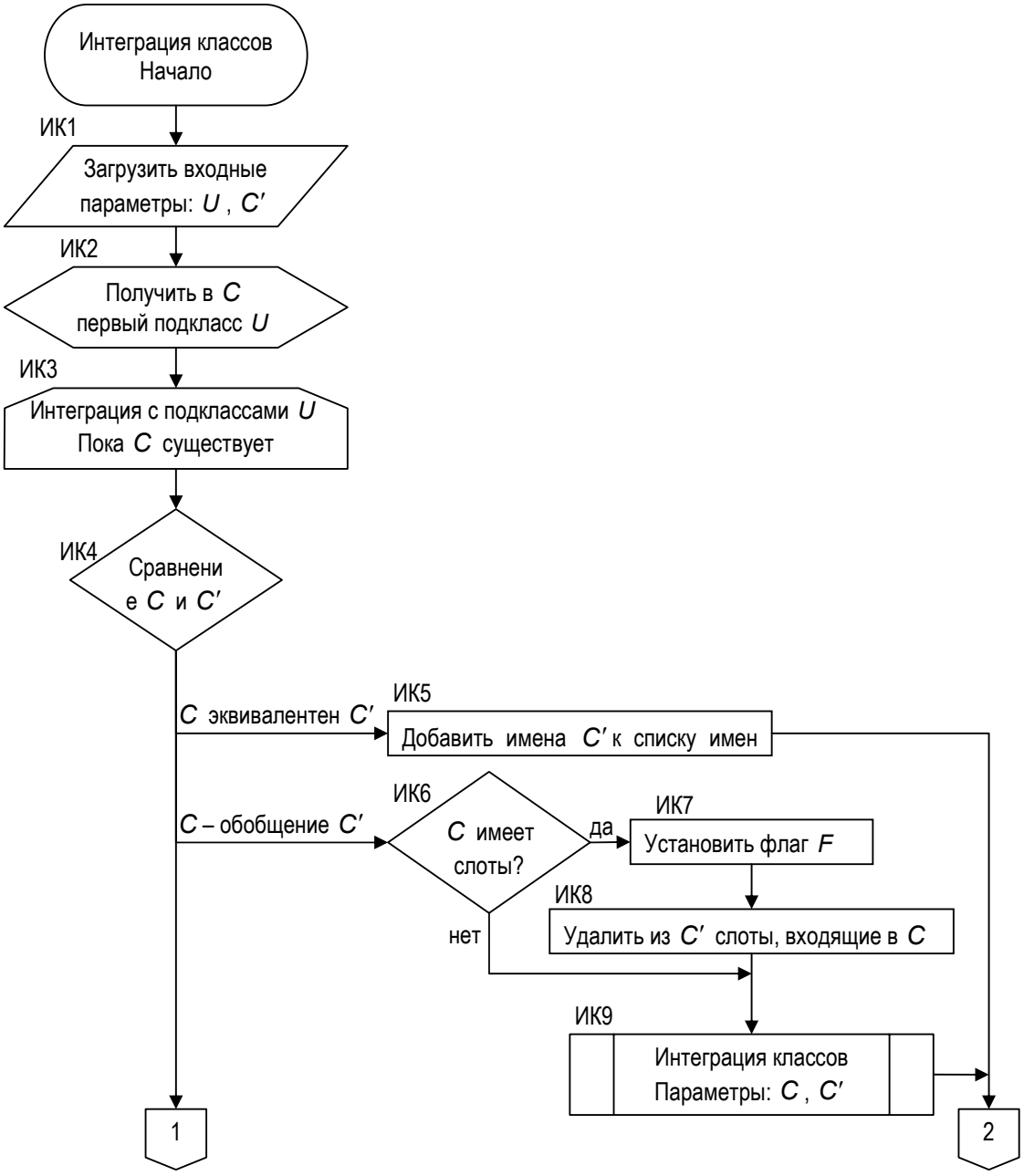

Рис. 2.в. Интеграция классов (начало)

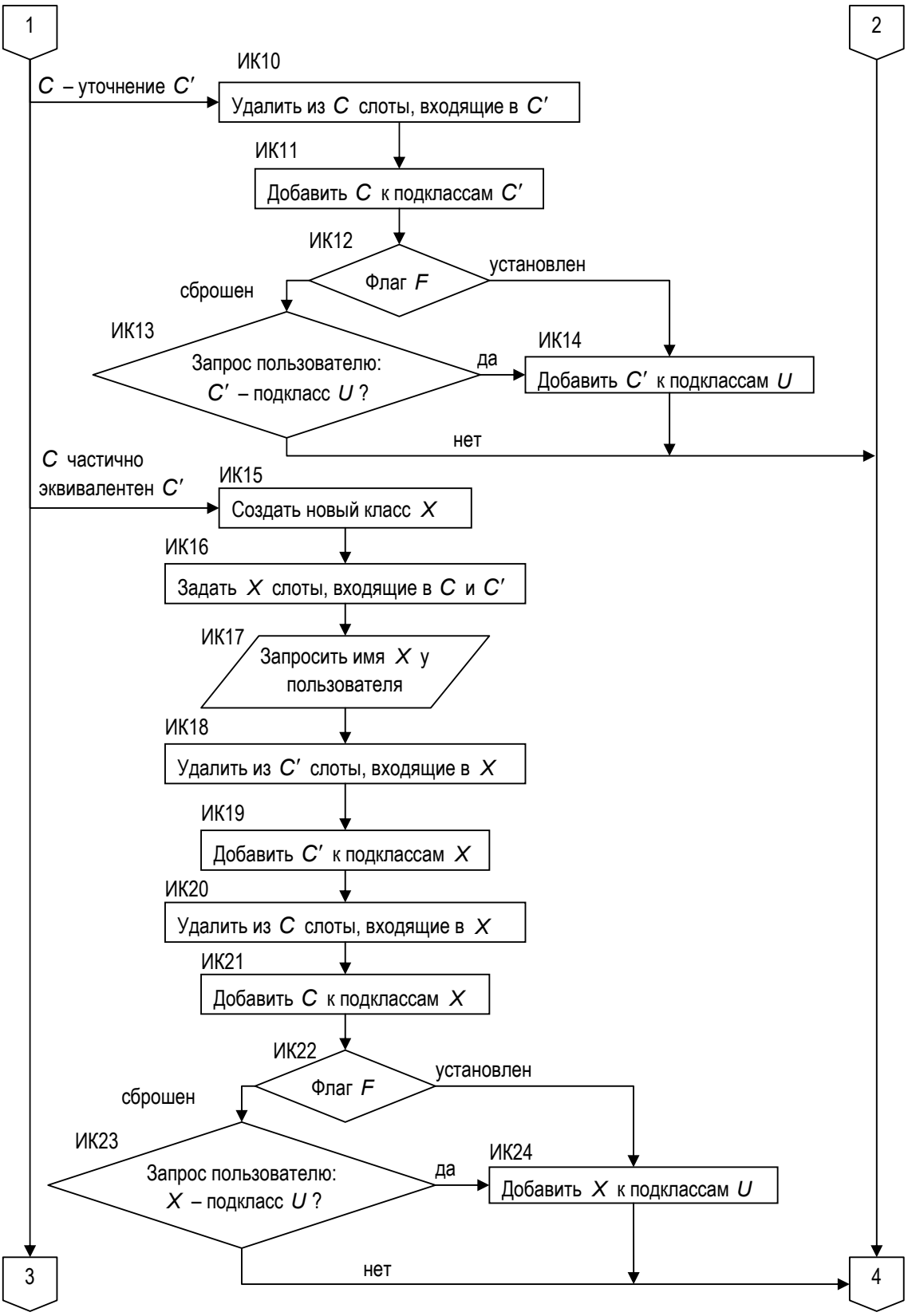

Рис. 2.в. Интеграция классов (продолжение)

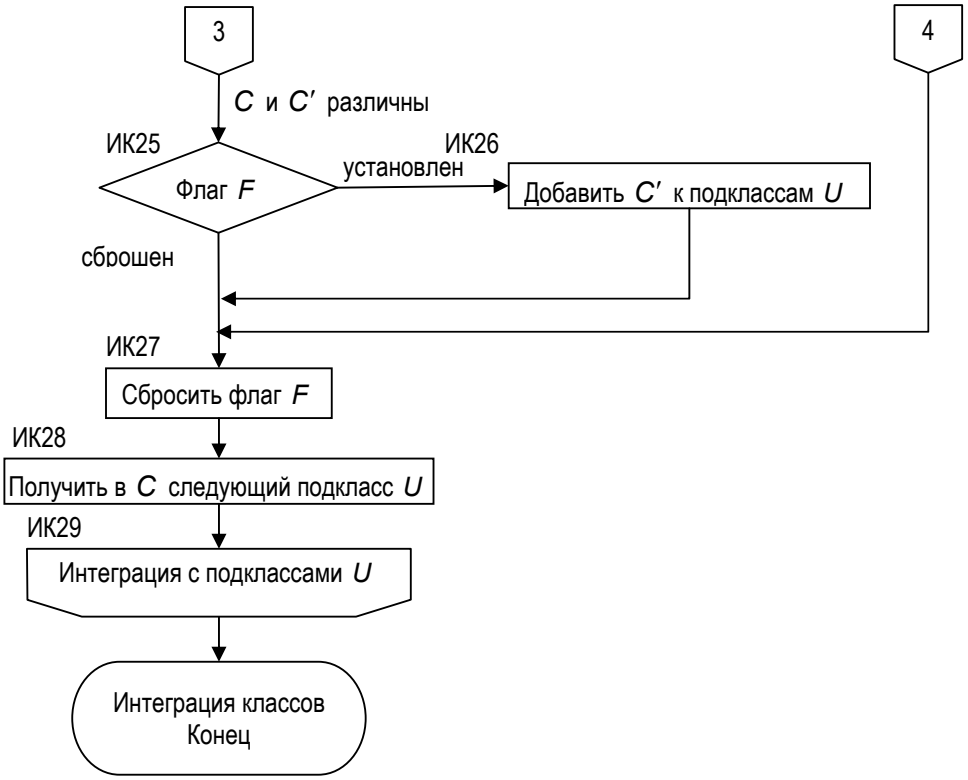

Рис. 2.в. Интеграция классов (окончание)

ИК1. Процедура «Интеграция классов» загружает входные параметры – классы U, C'. Класс U задает вершину родовидового дерева базовой онтологии, в которую будет интегрироваться класс C'.

ИК2. Задание временной переменной C (итератора для прохождения по списку непосредственных подклассов U) начального значения – первого непосредственного подкласса класса U в списке подклассов класса U.

ИК3. Начало цикла с предусловием. В цикле осуществляется сравнение и взаимное размещение классов C и C'. Цикл выполняется, пока не будет достигнут конец списка подклассов класса U, то есть подкласс C как элемент списка «удачно» извлечен (предполагается, что при достижении конца списка операция извлечения следующего его элемента вернет несуществующий «нулевой» элемент).

ИК4. Сравнение классов C и C' согласно механизму, описанному в подпункте «Базовый алгоритм сравнения классов».

ИК5. Если классы C и C' эквивалентны, к списку имен класса C добавляются имена C'. Иными словами, данный блок реализует объединение синонимов. Далее алгоритм переходит к завершающей части цикла (блоку **ИК27**).

ИК6. Если C является обобщением C', класс C' должен находиться ниже по иерархии относительно класса C. Однако, это может быть неверно, если C является «возможным обобщением» C', то есть принадлежит категориальному уровню и не содержит слотов. Осуществляется проверка на наличие слотов у класса C.

ИК7. Если C имеет слоты, значит C' должен быть добавлен в данную ветвь иерархии. Для учета этого при дальнейшем сравнении устанавливается флаг F (глобальная переменная логического типа).

ИК8. Из множества слотов класса C' удаляются слоты, входящие также в класс C, поскольку C' будет добавлен в подклассы класса C (непосредственно или через другие классы) на следующих этапах работы алгоритма, а значит, унаследует данные слоты.

ИК9. Независимо от того, содержит ли класс C слоты, необходимо сравнить класс C' с подклассами C. Для этого рекурсивно вызывается процедура «Интеграция классов» относительно C и C', реализуя тем самым «спуск» класса C' вниз по иерархии, заданной входящим классом U. Далее алгоритм переходит к завершающей части цикла (блок **ИК27**).

ИК10. Если класс C является уточнением класса C', класс C' должен находиться выше по иерархии относительно C. Из множества слотов класса C удаляются слоты, входящие также в класс C', поскольку класс C унаследует их от класса C'.

ИК11. Класс C добавляется в список подклассов класса C'.

ИК12. Проверка флага F, показывающего, была ли установлена в блоках **ИК6**-**ИК8** необходимость добавления класса C' в иерархию класса U (флаг установлен), или «спуск» класса C' производился через категориальные классы, которые не обязательно являются надклассами C' (флаг сброшен).

ИК13. Если флаг F сброшен, система не может принять однозначного решения относительно добавления класса C' в иерархию класса U: нет никаких признаков, что C' входит в иерархию C, но нет и никаких признаков, что C' не входит в иерархию U. Данную неоднозначность предлагается решить пользователю, для чего формируется вопрос: «Является ли C' подклассом U?». Если ответ пользователя отрицательный, алгоритм переходит к завершающей части цикла (блок **ИК27**).

ИК14. Если флаг F установлен, либо пользователь указал, что C' является подклассом U, класс C' добавляется в список подклассов класса U. Далее алгоритм переходит к завершающей части цикла (блок **ИК27**).

ИК15. Если C частично эквивалентен C', необходимо создать для классов C и C' общий надкласс, содержащий в себе слоты, входящие в оба сравниваемых класса. Создается класс X.

ИК16. В новый класс X копируются все слоты, общие для классов C и C'.

ИК17. Поскольку X не входил ни в одну из исходных онтологий, он не имеет имени. Система запрашивает имя класса X у пользователя, поясняя, что X – это обобщение классов C и C'.

ИК18. Из класса C' удаляются слоты, входящие в класс X, так как C', будучи подклассом X, их унаследует.

ИК19. Класс C' добавляется в список подклассов класса X.

ИК20. Из класса C удаляются слоты, входящие в класс X, так как C, будучи подклассом X, их унаследует.

ИК21. Класс C добавляется в список подклассов класса X.

ИК22. Аналогично блоку **ИК12**, устанавливается, как взаимно расположены классы X и U. Для этого проверяется флаг F.

ИК23. Если флаг F сброшен, система не может принять однозначного решения относительно добавления класса X в иерархию класса U. Данную неоднозначность предлагается решить пользователю, для чего формируется вопрос: «Является ли X подклассом U?». Если ответ пользователя отрицательный, алгоритм переходит к завершающей части цикла (блок **ИК27**).

ИК24. Если флаг F установлен, либо пользователь указал, что X является подклассом U, класс X добавляется в список подклассов класса U. Далее алгоритм переходит к завершающей части цикла (блок **ИК27**).

ИК25. Если классы C и C' различны, следует определить взаимное расположение классов C' и U проверкой флага F. Заметим, что в данном случае, если флаг F сброшен, нет оснований для предположений о взаимной связи классов C' и U, а различие C и C' дает возможность предположить обратное. Поэтому, если флаг F сброшен, алгоритм переходит к завершающей части цикла (блок **ИК27**).

ИК26. Если флаг F установлен, класс C' добавляется к списку подклассов U.

ИК27. Завершающая часть цикла – возможно, установленный флаг F сбрасывается.

ИК28. Итерирование по списку подклассов класса U – получение следующего элемента списка подклассов класса U и запись его в переменную C.

ИК29. Переход на начало цикла (блок **ИК3**) «Интеграция с подклассами U» с текущим значением итератора C.

Пример работы алгоритма. Пусть дано две онтологии, описывающие основные современные программно-аппаратные платформы персональных компьютеров. Базовая онтология (рис. 4) содержит следующие онтологические знания:

– существует две основных архитектуры процессоров: архитектура «RISC» и архитектура «x86»;

– существует два класса процессоров, отличающихся своей архитектурой: процессоры «PowerPC» с архитектурой «RISC» и процессоры «x86-совместимые» с архитектурой «x86»;

– существует класс компаний, производящих операционные системы: компании корпорации Microsoft (класс «Microsoft Corp.») и компании корпорации Apple (класс «Apple Inc.»);

– существует класс операционных систем, подразделяющийся на системы семейства «Windows» и «Mac OS», причем известно, что Windows-системы поддерживают базовые процессоры класса «x86-Совместимые» и производятся компаниями корпорации Microsoft, а системы Mac OS ориентированы на процессоры RISC и разрабатываются компаниями корпорации Apple.

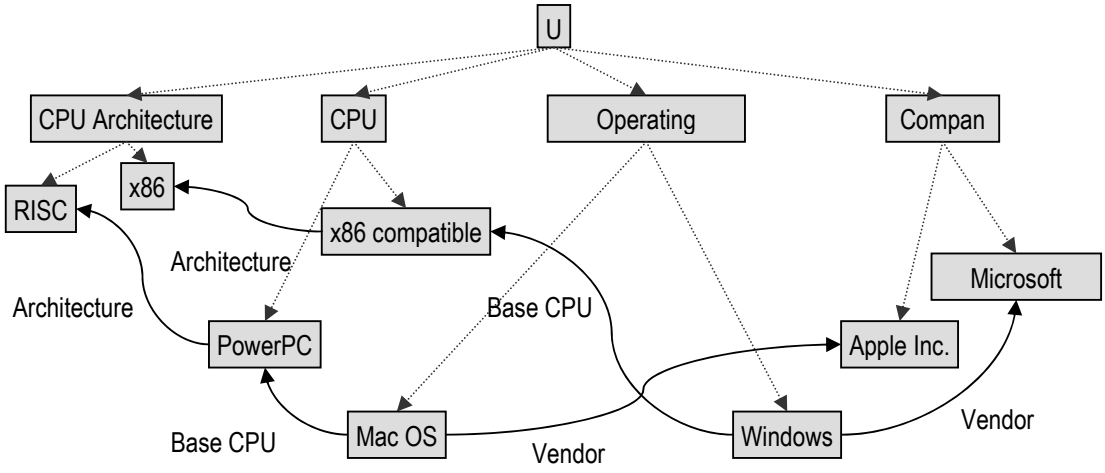

Рис. 4. Пример базовой онтологии

При поиске последней информации в данной ПдО поисковая система обнаружила и передала в модуль системной интеграции следующие онтологические знания в качестве интегрируемой онтологии (рис. 5):

– существует класс процессоров с архитектурой «x86», называемых семейством процессоров «Intel»;

– существует класс так называемых «Intel-based» операционных систем, поддерживающих базовые процессоры класса «Intel»;

– корпорацией Apple была разработана новая операционная система «Mac OS X» (точнее, класс операционных систем), принадлежащая классу «Intel-based» операционных систем.

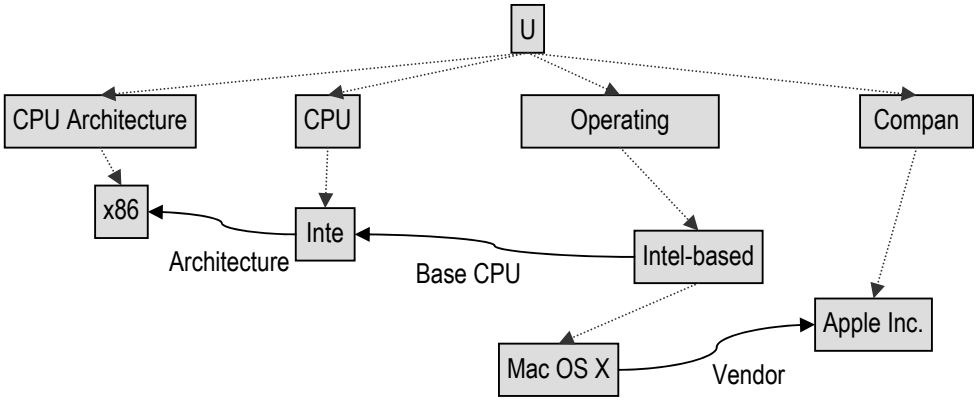

Рис. 5 Пример интегрируемой онтологии

В результате выполнения процедуры «Интеграция онтологий» базовая онтология будет дополнена онтологическими знаниями интегрируемой онтологии, изменив при этом и собственную исходную структуру, как это показано на рис. 6.

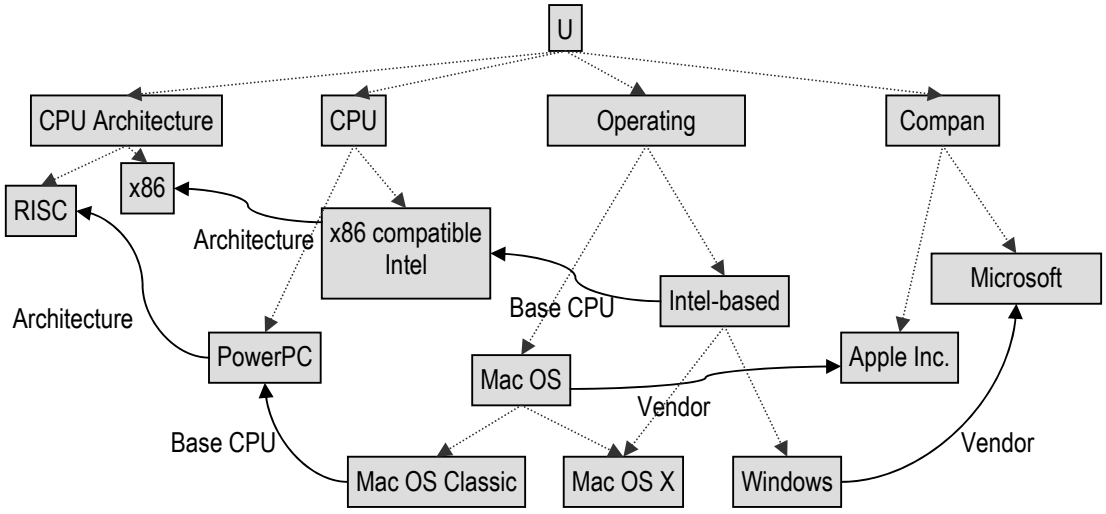

Рис. 6. Пример результирующей онтологии

Сравнение результатов работы алгоритма с результатами работы PROMPT. Произведя объединение двух данных онтологий средствами PROMPT, можно убедиться в значительных недостатках алгоритма его работы (см. рис. 7). Для объективной оценки алгоритма работы рассматриваемого плагина действия пользователя сводились к применению всех операций, предложенных PROMPT в том порядке, в котором они поступали, эмулируя тем самым автоматический режим работы.

Первым недостатком PROMPT является порядок определения пар классов для объединения или копирования – список операций в PROMPT строится в порядке добавления классов-аргументов операции в исходные онтологии. Очевидно, данный порядок является порядком хранения классов в онтологиях, то есть PROMPT не пытается предложить какую-либо «разумную» последовательность операций – он

выстраивает их «as is». Это, в свою очередь, привело к тому, что слоты в результирующую онтологию были добавлены до того, как были добавлены классы, на которые эти слоты ссылаются, что нарушило семантические связи между классами (на рис. 7 полностью отсутствуют семантические отношения).

Вторым недостатком PROMPT следует отметить способ сравнения классов. Несмотря на совпадение по содержанию классов «x86-compatible» и «Intel», они были скопированы в результирующую онтологию, так как имеют различные имена. Более того, в результирующей онтологии классы «Mac OS» и «Mac OS X» объединились по причине лексического подобия имен, что, несомненно, не соответствует действительности, так как данные операционные системы представляют абсолютно разные классы, отличающиеся типом базового процессора, что явно указано в исходных онтологиях.

Так же, не была установлена родовидовая связь между классами «Mac OS», «Mac OS X» и «Windows», так как PROMPT проводит сравнение классов с двумя вариантами результата сравнения («классы эквивалентны» или «классы различны») и не представляет никаких операций над классами, кроме объединения и копирования.

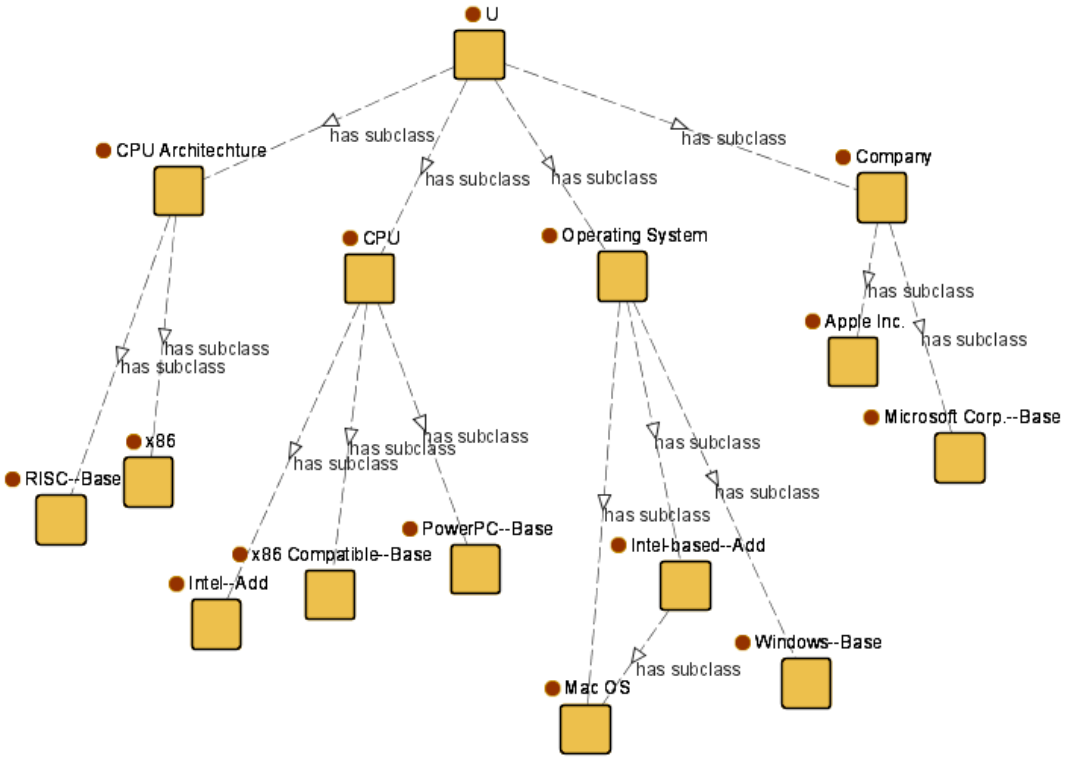

Рис. 7. Пример результирующей онтологии, полученной при помощи Protégé PROMPT

Выводы

Рассмотрение содержания класса как его определяющей составляющей позволяет обобщить операцию сравнения классов от определения классической эквивалентности (варианты «классы эквивалентны» и «классы различны») до установления родовидовых отношений, таких как «классы C_1 и C_2 эквивалентны», «класс C_1 является обобщением класса C_2», «класс C_1 является уточнением класса C_2», «классы C_1 и C_2 частично эквивалентны» и «классы C_1 и C_2 различны»). Обобщенная операция сравнения классов, в свою очередь, расширяет набор возможных при объединении онтологий действий от двух («объединить» для эквивалентных классов и «скопировать» – для различных) до пяти («объединить», «создать пару класс-подкласс», «создать пару класс-надкласс», «создать общий надкласс

для данных двух классов» и «скопировать классы» в соответствии с результатами сравнения). Такой расширенный набор операций над сравниваемыми классами, вместе с оригинальным алгоритмом перебора, сравнения и выполнения операций над классами, качественно усовершенствуют процедуру объединения онтологий, реализованную в современных онторедакторах, превращая ее в процедуру системной интеграции онтологий предметных областей.

Дальнейшие исследования в области интеграции онтологий ПдО следует посвятить усовершенствованию операции сравнения классов путем введения в сравнение сложных функций интерпретации (элементов множества F модели $\langle X, R, F \rangle$), использованию дополнительной информации об объемах классов, привлечению статистической информации о подобии классов, составленной на основе сравнения классов по содержанию. Целью исследований в этом направлении является приближение операции сравнения классов к уровню обработки глубинной семантики понятий, формирующих онтологии, тогда как полное достижение этого уровня является не разрешенной на сегодняшний день научной проблемой, что, очевидно, выходит за рамки данного исследования. Относительно процедуры интеграции онтологий, усовершенствования могут быть внесены в части преодоления возможных противоречий, вызванных введением в сравнение классов сложных функций интерпретации, визуализации процесса интеграции, предоставив пользователю возможность интерактивно следить за процессом, ускорения работы процедуры интеграции за счет оптимизации ее алгоритма. Сама разработанная процедура интеграции онтологий и алгоритм ее работы, описанные в данной статье, требуют дальнейшей экспериментальной проверки.

Литература

[Chaudhri 1998] V. Chaudhri, A. Farquhar, R. Fikes P. Karp J. Rice. OKBC: A Programmatic Foundation for Knowledge Base Interoperability. // Fifteenth National Conf. on Artificial Intelligence. AAAIPres/The MIT Press, Madison, P.600-607, 1998.

[Dean 2004] Mike Dean and Guus Schreiber. OWL Web Ontology Language Reference. W3C Recommendation. – 2004.

[Download Protégé] http://protege.stanford.edu/download/download.html

[Ehrig 2005] Marc Ehrig, York Sure. FOAM – Framework for Ontology Alignment and Mapping; Results of the Ontology Alignment Initiative // Proceedings of the Workshop on Integrating Ontologies, CEUR-WS.org, 156. – 2005. pp. 72-76.

[ISO 5807:1985] ISO 5807:1985. Information processing – Documentation symbols and conventions for data, program and system flowcharts, program network charts and system resources charts. International Organization for Standardization, 1985. – 25 p.

[Knublauch 2004] Holger Knublauch, Ray W. Fergerson, Natalya F. Noy, Mark A. Musen. The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications // Third International Semantic Web Conference – ISWC 2004, Hiroshima, Japan. – 2004. pp. 229-243.

[McGuinness 2000] McGuinness D., Fikes R., Rice J., Wilder S. An environment for merging and testing large ontologies // In Proc. of the Seventh Int. Conf., KR2000, Morgan Kaufmann Publishers, San Francisco, CA. – 2000. pp. 483-493.

[Noy 2000 (1)] Natalya Fridman Noy, Ray W. Fergerson, Mark A. Musen. The Knowledge Model of Protégé-2000: Combining Interoperability and Flexibility // Proceedings of the 12th European Workshop on Knowledge Acquisition, Modeling and Management, Springer-Verlag London, UK. – 2000. pp. 17-32.

[Noy 2000 (2)] Noy, N.F. and Musen, M.A. PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. // Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000), Austin, TX. – 2000. pp 450-455.

[Noy 2001] N. F. Noy, M. A. Musen. Anchor-PROMPT: Using Non-Local Context for Semantic Matching. // Workshop on Ontologies and Information Sharing at the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-2001), Seattle, WA. – 2001.

[Noy 2003] N.F. Noy and M.A. Musen The PROMPT Suite: Interactive Tools For Ontology Merging And Mapping // International Journal of Human-Computer Studies, 59/6. – 2003. pp. 983-1024.

[Protégé project] Stanford Centre for Biomedical Informatics Research: Protégé project publications. http://bmir.stanford.edu/publications/project.php/protg

[Sachs 2006] Eliza Sachs. Getting Started with Protégé-Frames. – 2006 – 72 p.

[Войшвилло 2007] Войшвилло, Е.К. Понятие как форма мышления: логико-гносеологический анализ. М.: Изд-во ЛКИ, 2007. – 240 с.

[ГОСТ 19.701-90] ГОСТ 19.701-90. Схемы алгоритмов, программ, данных и систем. Москва: Государственный комитет СССР по управлению качеством продукции и стандартам, 1990.

[Ивлев 2001] Ивлев Ю.В. Логика. Учебник для вузов. 2-е изд., перераб. и доп. М.: Логос, 2001. – 272 с.

[Овдей 2004] Овдей О.М., Проскудина Г.Ю. Обзор инструментов инженерии онтологий // Электронные библиотеки – Москва: Институт развития информационного общества, т.7 вып.4, 2004. – Электронный журнал, посвященный созданию и использованию электронных библиотек. http://www.elbib.ru/

[Палагін 2006] Палагін О.В., Петренко М.Г. Модель категоріального рівня мовно-онтологічної картини світу // Математичні машини і системи. – 2006. – №3. – С. 91-104.

[Палагін 2009] Палагін О., Кургаєв О. Міждисциплінарні наукові дослідження: оптимізація системно-інформаційної підтримки // Вісник НАН України. – 2009. – № 3. – С. 14–25.

Информация об авторах

Палагин Александр Васильевич *– Ин-т кибернетики им. В.М. Глушкова НАН Украины, Киев-187 ГСП, 03680, просп. акад. Глушкова, 40; e-mail: palagin_a@ukr.net*

Основные области научных исследований: системная интеграция трансдисциплинарных научных знаний, онтологический инжиниринг

Андрей Васильевич Михайлюк *– Ин-т кибернетики им. В.М. Глушкова НАН Украины, Киев-187 ГСП, 03680, просп. акад. Глушкова, 40; e-mail: fruler@ukr.net*

Основные области научных исследований: формальные модели представления знаний, логико-онтологические системы обработки знаний

Величко Виталий Юрьевич *– Ин-т кибернетики им. В.М. Глушкова НАН Украины, Киев-187 ГСП, 03680, просп. акад. Глушкова, 40; e-mail: velychko@aduis.com.ua*

Основные области научных исследований: индуктивный логический вывод, обработка естественно-языковых текстов.

Петренко Николай Григорьевич *– Ин-т кибернетики им. В.М. Глушкова НАН Украины, Киев-187 ГСП, 03680, просп. акад. Глушкова, 40; e-mail: petrng@ukr.net*

Основные области научных исследований: методология и инструментальные средства автоматизированного проектирования онтологий предметных областей, системная интеграция трансдисциплинарных научных знаний