№1. Проектирование и разработка веб-онтологий

Цель практической работы – изучение основных приемов и методов инженерии веб-онтологий на языке OWL на основе инструментального программного средства Protégé.

Упражнения

Упражнение 1. Запустите редактор Protégé.

1. Запустите программу из меню Пуск – Программы – Protégé.

Упражнение 2. Создайте новый проект.

1. Выберите вкладку *Create New Project (Создать новый проект)*.
2. В диалоге *Create New Project* выберите тип проекта *OWL Files (.owl or .rdf)* и нажмите кнопку *Finish*.

Классы

Класс определяет группу индивидов, которых объединяет наличие некоторых общих свойств. Классы могут быть организованы в иерархии. Самый общий класс *owl:Thing* является классом всех индивидов и суперклассом для всех классов.

Упражнение 3. Создайте классы *Pizza (Пицца)*, *PizzaTopping (Начинка пиццы)* и *PizzaBase (Основа пиццы)* как подклассы *owl:Thing*.

1. Выделите класс *owl:Thing* в окне *Subclass Relationship (Взаимоотношения подклассов)* и нажмите на иконке  *Create subclass (Создать подкласс)*. При этом новый класс будет создан как подкласс выделенного класса.
2. Измените название класса на *Pizza* в окне *Class editor (Редактор классов)* и нажмите клавишу ввода.
3. По аналогии создайте классы *PizzaTopping* и *PizzaBase* как подклассы *owl:Thing*.

Упражнение 4. Преобразуйте классы *Pizza*, *PizzaTopping* и *PizzaBase* в независимые.

1. Выделите класс *Pizza* в иерархии классов.
2. Нажмите на иконке  *Add all siblings (Добавить все сестринские элементы)* в окне *Disjoints (Независимые классы)* в правом нижнем углу экрана.
3. В диалоге *Add siblings to disjoints (Сделать сестринские элементы независимыми)* выберите *Mutually between all siblings (Взаимно между всеми сестринскими элементами)* и нажмите *OK*. Убедитесь, что классы *PizzaBase* и *PizzaTopping* присутствуют в списке независимых для класса *Pizza*, классы *Pizza* и *PizzaBase* – для класса *PizzaTopping*, классы *Pizza* и *PizzaTopping* – для класса *PizzaBase*.

Упражнение 5. Используйте инструмент создания множественных подклассов для создания классов *ThinAndCrispy* и *DeepPan* как подклассов *PizzaBase.*

1. Выделите класс *PizzaBase* в иерархии классов.
2. Выберите *Quick OWL – Create multiple subclasses (Создать множественные подклассы)* из пункта *Tools (Инструменты)* главного меню. Нажмите кнопку *Next*.
3. В строке 1 введите название подкласса *ThinAndCrispy* и нажмите клавишу ввода. Программа проверит имя на уникальность и соответствие используемой нотации. Ошибки выводятся в статус-баре в нижней части экрана. В строке 2 введите название подкласса *DeepPan*. По завершению нажмите кнопку *Next*.
4. Убедитесь, что установлена галочка *Make all primitive siblings disjoint* (*Сделать все сестринские элементы независимыми)*. Нажмите кнопку *Finish*.

Упражнение 6. Используйте инструмент создания множественных подклассов для создания подклассов *PizzaTopping.*

1. Выделите класс *PizzaTopping* в иерархии классов.
2. По аналогии с упражнением 5 создайте подклассы в соответствии с рис. 1. Для программирования иерархии третьего и четвертого уровня

используйте клавишу табуляции. Убедитесь, что подклассы являются независимыми между собой.

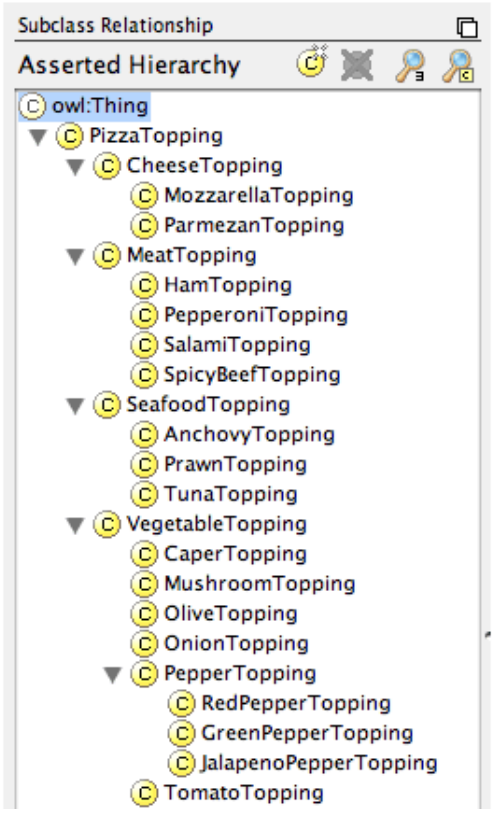

Рис. 3.5. Иерархия подклассов PizzaTopping

Сохраните проект.

1. Выберите *Save Project As…* из пункта *File* главного меню.
2. В поле *Project* укажите *Pizza-ГРУППА-ФАМИЛИЯ_1-ФАМИЛИЯ_2*, например, *Pizza-M41-Ivanova-Petrova*. Нажмите кнопку *OK*. В дальнейшем регулярно сохраняйте проект, выбирая *Save Project* из пункта *File* главного меню.

Свойства

Свойства предназначены для того, чтобы установить отношения между индивидами (свойства-объекты) или от индивидов к значениям данных (свойства-значения).

Рассмотрим свойства *имеетРебенка*, *имеетРодственника*, *имеетБрата* и *имеетВозраст*. Первые три могут использоваться, чтобы связать представителя класса *Человек* с другими представителями класса

Человек (и, таким образом, являются свойствами-объектами), последнее свойство может использоваться, чтобы связать представителя класса *Человек* с представителем типа данных *Целое число* (и, таким образом, является свойством-значением).

Упражнение 7. Создайте свойство *hasIngredient.*

1. Переключитесь на вкладку *Properties (Свойства)*. Нажмите на иконке  *Create object property (Создать свойство-объект)*.
2. В окне *Property editor (Редактор свойств)* укажите название свойства *hasIngredient* и нажмите клавишу ввода.

Упражнение 8. Создайте свойства *hasTopping* и *hasBase* как подсвойства *hasIngredient.*

1. Нажмите правой кнопкой мыши на свойстве *hasIngredient* в иерархии свойств.
2. Выберите *Create subproperty (Создать подсвойство)*. При этом новое свойство будет создано как подсвойство *hasIngredient*.
3. Укажите название свойства *hasTopping* в окне *Property editor* и нажмите клавишу ввода.
4. По аналогии создайте свойство *hasBase* как подсвойство *hasIngredient*.

Одно свойство может быть представлено как инверсия другого. Если свойство *P1* представлено как инверсия свойства *P2*, то, если *X* связан с *Y* свойством *P2*, то *Y* связан с *X* свойством *P1*.

Например, если свойство *имеетРебенка* – инверсия свойства *имеетРодителя*, а Дебора *имеетРодителя* Луиза, можно вывести, что Луиза *имеетРебенка* Дебора.

Упражнение 9. Создайте свойство *isIngredientOf* как инверсию свойства *hasIngredient*, свойство *isBaseOf* как инверсию свойства *hasBase* и свойство *isToppingOf* как инверсию свойства *hasTopping*.

1. По аналогии с упражнением 7 создайте свойство *isIngredientOf* и выделите его в иерархии свойств.

2. Нажмите на иконке  *Set inverse property (Задать свойство инверсии)* в окне *Inverse*. В диалоге выберите свойство *hasIngredient* и нажмите кнопку *OK*. Убедитесь, что свойство *hasIngredient* появилось в окне *Inverse*. При этом в иерархии свойств будет отражено, что свойства *hasIngredient* и *isIngredientOf* являются инверсией.
3. Выделите свойство *hasBase* в иерархии свойств.
4. Нажмите на иконке  *Create new inverse property (Создать новое свойство инверсии)* в окне *Inverse*. Укажите название свойства *isBaseOf* и закройте окно диалога. Обратите внимание, что свойство *isBaseOf* было создано как подсвойство *hasIngredientOf*. Это согласуется с тем, что свойство *hasBase* является подсвойством *hasIngredient*, а свойство *isIngredientOf* является инверсией для *hasIngredient.*
5. Выделите свойство *hasTopping* в иерархии свойств.
6. Создайте свойство *isToppingOf* как инверсию свойства *hasTopping*.

Свойства могут быть объявлены транзитивными. Если пара *(x,y)* – представитель транзитивного свойства *P* и пара *(y,z)* – представитель свойства *P*, то пара *(x,z)* – также представитель *P*.

Например, если свойство *Предок* объявлено транзитивным, и если Сара – предок Луизы (т.е. (Сара,Луиза) – представитель свойства Предок), а Луиза – предок Деборы (т.е. (Луиза,Дебора) – представитель свойства Предок), то можно вывести, что Сара является предком Деборы (т.е. (Сара,Дебора) – представитель свойства *Предок*).

Упражнение 10. Объявите свойство *hasIngredient* транзитивным.

1. Выделите свойство *hasIngredient* в иерархии свойств.
2. Поставьте галочку напротив *Transitive (Транзитивное)*.
3. Выделите свойство *isIngredientOf*, являющееся инверсией свойства *hasIngredient*.
4. Поставьте галочку напротив *Transitive*.

Свойства могут быть заданы как имеющие уникальное значение. Если свойство объявлено функциональным, то оно имеет не более одного значения для каждого индивида.

Пусть свойство *имеетГлавногоРаботодателя* объявлено функциональным. Из этого можно вывести, что никакой индивид не может иметь более чем одного главного работодателя. Что, впрочем, не подразумевает, что каждый индивид должен иметь, по крайней мере, одного главного работодателя.

Упражнение 11. Объявите свойство *hasBase* функциональным, а свойство *isBaseOf* – обратно-функциональным.

1. Выделите свойство *hasBase* в иерархии свойств.
2. Поставьте галочку напротив *Functional (Функциональное)*.

Если свойство обратно-функционально, то инверсия этого свойства функциональна. То есть инверсия данного свойства имеет не более одного значения для каждого индивида.

Пусть свойство *имеетИНН* (идентификационный номер налогоплательщика) объявлено как обратно-функциональное. Инверсия этого свойства *служитИННдля* имеет не более одного значения для любого индивида в классе номеров налогоплательщиков. То есть единственное значение свойства *служитИННдля* для конкретного номера налогоплательщика – это ИНН какого-либо человека. Иными словами, нет двух различных представителей в классе *Человек* с одним и тем же ИНН. Кроме того, можно вывести что, если два представителя в классе Человек имеют один и тот же ИНН, то эти два представителя ссылаются на одного и того же индивида.

3. Осуществите проверку онтологии, для чего выберите *Run ontology test (Запустить тест онтологии)* из пункта *OWL* главного меню. Выделите сообщение об ошибке и нажмите на иконке *Repair selected item (Исправить выбранный пункт)*.

4. Выделите свойство *isBaseOf* и убедитесь, что была автоматически проставлена галочка напротив *InverseFunctional (Обратно-функциональное)*.

Диапазон свойства ограничивает индивидов, которые могут выступать в качестве значений этого свойства. Если свойство связывает одного индивида с другим, и это свойство имеет класс в качестве своего диапазона, то другой индивид должен принадлежать этому классу.

Например, для свойства *имеетРебенка* может быть заявлен диапазон *Млекопитающие*. Из этого можно вывести что, если Луиза связана с Деборой свойством *имеетРебенка*, (т.е. Дебора – ребенок Луизы), то Дебора относится к *Млекопитающим*. Диапазон как и домен – глобальные ограничения, так как они накладываются на все свойство, а не только на свойство, связанное с конкретным классом.

Упражнение 12. Определите *PizzaTopping* в качестве диапазона для свойства *hasTopping*.

1. Выделите свойство *hasTopping* в иерархии свойств.
2. Нажмите на иконке  *Add named class (Добавить именованный класс)* в окне *Range (Диапазон свойства)*.
3. В появившемся диалоге выберите класс *PizzaTopping* и нажмите кнопку *OK*. Убедитесь, что свойство *PizzaTopping* появилось в списке диапазонов.

Домен (область определения) свойства ограничивает индивидов, к которым это свойство может быть применено. Если свойство связывает индивида с другим индивидом, и это свойство имеет какой-то класс в качестве одного из своих доменов, то индивид должен принадлежать к этому классу.

Например, свойство *имеетРебенка* имеет область определения *Млекопитающие*. Из этого можно вывести что, если Франк *имеетРебенка* Анна, то Франк – *Млекопитающее*.

Упражнение 13. Определите *Pizza* в качестве домена для свойства *hasTopping*.

1. Выделите свойство *hasTopping* в иерархии свойств.
2. Нажмите на иконке  *Add named class (Добавить именованный класс)* в окне *Domain (Домен свойства)*.
3. Появится диалог, который позволяет выбрать класс в иерархии классов. Выберите класс *Pizza* и нажмите кнопку *OK*. Убедитесь, что свойство *Pizza* появилось в списке доменов.

Упражнение 14. Задайте домен и диапазон для свойства *isToppingOf* и его инверсии *hasTopping*.

1. Убедитесь, что домен и диапазон для инверсии *isToppingOf* были определены автоматически на основе домена и диапазона свойства *hasTopping*.

Упражнение 15. Задайте домен и диапазон значений для свойства *hasBase* и его инверсии *isBaseOf*.

1. Выделите свойство *hasBase*.
2. Определите *PizzaBase* в качестве диапазона для *hasBase*.
3. Определите *Pizza* в качестве домена для *hasBase*.
4. Убедитесь, что домен и диапазон для инверсии *isBaseOf* были определены автоматически на основе домена и диапазона свойства *hasBase*.

Ограничения свойств используются для ограничения идивидов, принадлежащих классу. Выделяют три типа ограничений: кванторные, ограничения кардинальности и ограничения на *hasValue*.

Кванторное ограничение состоит из квантора, свойства и значения слота. Кванторное ограничение существования (∃ *someValuesFrom*) подразумевает такое ограничение свойства, при котором хотя бы одно из значений этого свойства должно иметь определенный тип.

Например, класс *СтатьяСемантическойСети* может иметь ограничение *someValuesFrom* для свойства *имеетКлючевоеСлово*,

согласно которому некоторые значения свойства *имеетКлючевоеСлово* должны быть представителями класса *РубрикаСемантическойСети*. Чтобы статья была совместимой с ограничением *someValuesFrom,* среди множества ключевых слов должен быть один или несколько представителей класса *РубрикаСемантическойСети*.

Упражнение 16. Добавьте ограничение существования для *Pizza,* которое определяет, что каждая пицца должна иметь основу (*PizzaBase).*

1. Перейдите на вкладку *OWL Classes* и выделите класс *Pizza* в иерархии классов.
2. Выберите заголовок *NECESSARY (Необходимое)* в окне *Conditions/Asserted Conditions (Условия/Объявленные условия)*.
3. Нажмите на иконке  *Create restriction (Создать ограничение)*. Появится диалог *Create Restriction*, который мы будем использовать для создания ограничения.

Упражнение 17 (продолжение).

1. Выберите ограничение существования вида ∃ *someValuesFrom* в списке ограничений.
2. Выберите свойство *hasBase* в списке свойств.
3. Задайте *PizzaBase* в качестве значения слота. Для этого либо непосредственно введите *PizzaBase* в окне редактирования, либо нажмите на иконке  *Insert class (Вставить класс)* на панели построения выражений, чтобы вызвать дерево иерархий классов, где выберите *PizzaBase*.
4. Нажмите *OK*, чтобы создать ограничение и закрыть диалог. Убедитесь, что созданное ограничение появилось в окне *Conditions/Asserted Conditions*. В случае ошибки диалог не закроется, а в нижней части панели появится сообщение об ошибке.

Упражнение 18. Создайте класс *NamedPizza* в качестве подкласса для *Pizza* и класс *MargheritaPizza* в качестве подкласса для *NamedPizza.*

1. Выберите класс *Pizza* в иерархии классов на вкладке *OWL Classes*.
2. По аналогии с упражнением 3 создайте класс *NamedPizza* в качестве подкласса для *Pizza* и класс *MargheritaPizza* в качестве подкласса для *NamedPizza*.
3. Добавьте комментарий для класса *MargheritaPizza*: «A pizza that only has Mozarella and Tomato toppings.» (Пицца, содержащая только сыр «Моцарелла» и помидоры.) В хорошей онтологии классы, свойства и т.д. обязательно документируются.

Упражнение 19. Создайте на *MargheritaPizza* ограничение существования для свойства *hasTopping* со значением слота *MozzarellaTopping*, которое определяет, что *MargheritaPizza* имеет по меньшей мере одну сырную начинку (*MozzarellaTopping*).

1. Выберите класс *MargheritaPizza* в иерархии классов.
2. Выберите заголовок *NECESSARY* в окне *Conditions/Asserted Conditions*.
3. Нажмите на иконке  *Create restriction (Создать ограничение)*.
4. Выберите ограничение существования вида ∃ *someValuesFrom* в списке ограничений.
5. Выберите свойство *hasTopping* в списке свойств.
6. Выберите класс *MozzarellaTopping* в качестве значения слота для создаваемого ограничения.
7. Нажмите *OK*, чтобы создать ограничение и закрыть диалог.

Упражнение 20. Создайте на *MargheritaPizza* ограничение существования для свойства *hasTopping* со значением слота *TomatoTopping*, которое определяет, что *MargheritaPizza* имеет по меньшей мере одну томатную начинку (*TomatoTopping*).

1. Выберите класс *MargheritaPizza* в иерархии классов.
2. Создайте ограничение существования по аналогии с упражнением 19.

Упражнение 21. Создайте пиццу *AmericanaPizza*, используя клонирование и модификацию описания пиццы *MargheritaPizza*.

1. Нажмите правой кнопкой мыши на классе *MargheritaPizza* в иерархии классов.
2. Выберите *Create clone (Клонировать)*. Будет создана копия класса *MargheritaPizza* с названием *MargheritaPizza2*.
3. Переименуйте *MargheritaPizza2* в *AmericanaPizza*.
4. Выберите *AmericanaPizza* в иерархии классов.
5. По аналогии с упражнением 19 добавьте *PepperoniTopping* в качестве обязательного компонента *AmericanaPizza (∃ someValuesFrom, hasTopping, PepperoniTopping)*.

Упражнение 22. Создайте пиццу *AmericanHotPizza* и пиццу *SohoPizza.*

1. *AmericanHotPizza* отличается от *AmericanaPizza* наличием дополнительного компонента – перца чили (*JalapenoPepperTopping*). Создайте эту пиццу по аналогии с упражнением 21.
2. *SohoPizza* отличается от *MargheritaPizza* наличием двух дополнительных компонентов – оливок (*OliveTopping*) и сыра «Пармезан» (*ParmezanTopping*). Создайте эту пиццу по аналогии с упражнением 21.

Упражнение 23. Сделайте подклассы *NamedPizza* дизъюнктивными.

1. Выполняется по аналогии с упражнением 4.

Блок рассуждений

Блок рассуждений позволяет автоматически вычислять классификационные иерархии и проверять онтологии на логическую правильность.

Запустите блок рассуждений RACER.

Запустите программу RacerPro из меню Пуск – Программы – RacerPro и сверните ее в трей.

Упражнение 24. Создайте тестовый класс *ProbeInconsistentTopping*, являющийся подклассом одновременно для *CheeseTopping* и *Vegetable*.

1. Выберите класс *CheeseTopping* в иерархии классов.
2. Создайте класс *ProbeInconsistentTopping* как подкласс *CheeseTopping*.
3. Добавьте следующий комментарий для класса *ProbeInconsistentTopping*: «This class should be inconsistent when the ontology is classified.» (Этот класс будет противоречивым при классификации онтологии.)
4. Убедитесь, что класс *ProbeInconsistentTopping* выбран в иерархии классов, выделите заголовок *NECESSARY* в окне условий.
5. Нажмите на иконке *Add named class*. Появится диалог, содержащий иерархию классов. Выберите класс *VegetableTopping* и нажмите клавишу *OK*. Класс будет добавлен как необходимое условие.

Упражнение 25. Классифицируйте онтологию, чтобы убедиться, что класс ProbeInconsistentType является противоречивым.

1. Нажмите кнопку *Classify Taxonomy* из пункта *OWL* главного меню, чтобы классифицировать онтологию. В вычисленной (логически выведенной) иерархии класс *ProbeInconsistentType* будет взят в красный кружок как противоречивый. Вывод о том, что объект не может быть одновременно овощем и сыром, блок рассуждений сделал на основе дизъюнктивности подклассов *PizzaTopping*.

Упражнение 26. Удалите условие дизъюнктивности классов *CheeseTopping* и *VegetableTopping*, чтобы устранить противоречие.

1. Выберите класс *CheeseTopping* в иерархии классов. Окно *Disjoints* будет содержать *VegetableTopping, SeafoodTopping, MeatTopping.*
2. Выберите класс *VegetableTopping*. Нажмите на иконке *Delete selected row (Удалить текущую строку)*, чтобы исключить аксиому дизъюнктивности для классов *CheeseTopping* и *VegetableTopping*.

3. Нажмите кнопку *Classify Taxonomy* из пункта *OWL* главного меню, чтобы классифицировать онтологию. Убедитесь, что несоответствия для *ProbeInconsistentTopping* больше не существует.

Упражнение 27. Исправьте онтологию, сделав классы CheeseTopping и Vegetable снова дизъюнктивными.

1. Выберите класс *CheeseTopping* в иерархии классов.
2. Окно *Disjoints* будет содержать классы *SeafoodTopping, MeatTopping.*
3. Нажмите на иконке  *Add disjoint class (Добавить независимый класс)*. В окне диалога выберите класс *VegetableTopping* и нажмите *OK*. Классы *CheeseTopping* и *VegetableTopping* снова станут дизъюнктивными.
4. Классифицируйте онтологию, чтобы убедиться, что класс *ProbeInconsistentTopping* снова помечен как противоречивый.

Класс, содержащий только необходимые условия, называют примитивным. Необходимое условие означает: если объект является членом класса, то он обязательно подчиняется указанным условиям. Наличие только необходимого условия недостаточно, чтобы сказать: если объект выполняет указанные условия, значит, он будет членом класса.

Упражнение 28. Создайте класс *CheesyPizza* как подкласс для *Pizza* и задайте условие, согласно которому сырная пицца *CheesyPizza* содержит не менее одной сырной начинки *CheeseTopping*.

1. Выберите *Pizza* в иерархии классов.
2. Создайте подкласс *CheesyPizza*.
3. Выберите *CheesyPizza*.
4. По аналогии с упражнением 19 создайте на *CheesyPizza* ограничение существования для свойства *hasTopping* со значением слота *CheeseTopping*.

Упражнение 29. Преобразуйте необходимые условия для *CheesyPizza* в необходимые и достаточные.

1. Выберите *CheesyPizza* в иерархии классов.

2. На окне условий выберите ограничение ∃ *hasTopping CheeseTopping*.
3. Перетащите ограничение ∃ *hasTopping CheeseTopping* на заголовок *NECESSARY AND SUFFICIENT (Необходимое и достаточное)*.
4. На окне условий выберите класс *Pizza*.
5. Перетащите *Pizza* поверх ограничения ∃ *hasTopping CheeseTopping* (обратите внимание – не поверх заголовка).
6. Убедитесь, что результат соответствует рис. 2.

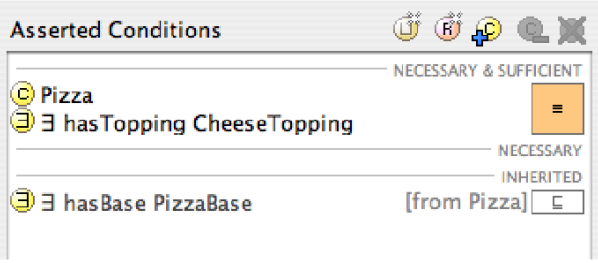

Рис. 3.6. Необходимые и достаточные условия

Упражнение 30. Используйте блок рассуждений для автоматического вычисления подклассов *CheesyPizza*.

1. Нажмите кнопку *Classify Taxonomy* из пункта *OWL* главного меню, чтобы классифицировать онтологию. Блок рассуждений поместил *MargheritaPizza*, *AmericanaPizza*, *AmericanHotPizza* и *SohoPizza* в подклассы *CheesyPizza*, потому что мы определили *CheesyPizza*, используя необходимые и достаточные условия.

Упражнение 31. Создайте класс, описывающий вегетарианскую пиццу *VegetarianPizza*.

1. Создайте класс *VegetarianPizza* как подкласс для *Pizza*.
2. Убедитесь, что класс *VegetarianPizza* выбран в иерархии классов и выделите заголовок *NECESSARY* в окне условий.
3. Нажмите на иконке *Create restriction (Создать ограничение)*.
4. Выберите универсальное ограничение (квантор общности) вида ∀ *allValuesFrom* в списке ограничений.
5. Выберите свойство *hasTopping* в списке свойств.

6. В качестве значения слота для создаваемого ограничения укажите *CheeseTopping* ИЛИ *VegetableTopping*. Для этого сначала выберите класс *CheeseTopping*, нажав на иконку *Insert class*, а затем вставьте оператор *unionOf*, нажав на соответствующую иконку. После этого выберите класс *VegetableTopping*, чтобы получить *CheeseTopping* ∪ *VegetableTopping*.
7. Нажмите *OK*, чтобы создать ограничение и закрыть диалог.

Упражнение 32. Преобразуйте необходимые условия для *VegetarianPizza* в необходимые и достаточные.

1. Выполняется по аналогии с упражнением 29.

Упражнение 33. Классифицируйте онтологию.

Нажмите кнопку *Classify Taxonomy* из пункта *OWL* главного меню, чтобы классифицировать онтологию. Обратите внимание, что *MargheritaPizza* и *SohoPizza* не классифицированы как подклассы *VegetarianPizza*, хотя и содержат сыр и овощи в качестве ингредиентов. Дело в том, что согласно логике Open World Reasoning, пока мы не скажем явно, что *MargheritaPizza* содержит только *MozzarellaTopping* и *TomatoTopping* в качестве начинки, блок рассуждений будет считать, что *MargheritaPizza* может иметь другие начинки. Такое явное высказывание можно задать с помощью соответствующей аксиомы.

Упражнение 34. Создайте аксиому ∀ *hasTopping (MozzarellaTopping* ∪ *TomatoTopping)* на свойстве *hasTopping* для *MargheritaPizza*.

- *MargheritaPizza*
- *NECESSARY*
- ∀ *allValuesFrom*
- *hasTopping*
- *MozzarellaTopping* ∪ *TomatoTopping*

Упражнение 35. Создайте аксиому ∀ *hasTopping (ParmezanTopping* ∪ *MozzarellaTopping* ∪ *TomatoTopping* ∪ *OliveTopping)* на свойстве *hasTopping* для *SohoPizza*.

1. Выполняется по аналогии с упражнением 34.

Упражнение 36. Автоматически создайте аксиому на свойстве *hasTopping* для *AmericanaPizza*.

1. Выберите класс *AmericanaPizza* в иерархии классов.
2. Нажмите правой кнопкой мыши на одном из ограничений существования *hasTopping* на вкладке *Conditons*. Выберите *Add closure axiom (Добавить аксиому)*. Будет создано универсальное ограничение (квантор общности) для свойства *hasTopping*, имеющее в качестве значения слота объединение всех существующих значений слотов.

Упражнение 37. Автоматически создайте аксиому на свойстве *hasTopping* для *AmericanHotPizza*.

Выполняется по аналогии с упражнением 36.

Упражнение 38. Классифицируйте онтологию.

1. Нажмите кнопку *Classify Taxonomy* из пункта *OWL* главного меню, чтобы классифицировать онтологию. Обратите внимание, что *MargheritaPizza* и *SohoPizza* классифицированы как вегетарианские (подклассы *VegetarianPizza*).
2. Подключите плагин OWLViz из меню Project – Configure. Перейдите на вкладку OWLViz и проанализируйте результат. При обнаружении ошибок исправьте их и заново классифицируйте онтологию.

Упражнение 39. Добавьте в качестве подкласса *NamedPizza* пиццу *Napoletana*, содержащую следующие компоненты: каперсы, анчоусы, помидоры, сыр «моцарелла», оливки.

Упражнение 40. Добавьте в качестве подкласса *NamedPizza* пиццу *FourSeasons*, содержащую следующие компоненты: пепперони, анчоусы, оливки, помидоры, каперсы, сыр «моцарелла», грибы.

Упражнение 41. Выделите класс пицц *MeatyPizza*, содержащих мясную начинку. Произведите классификацию.

Упражнение 42. Выделите класс пицц *InterestingPizza*, содержащих, как минимум, три вида начинок (ограничение вида $\geq \min$ *Cardinality* для свойства *hasTopping*). Произведите классификацию.

Упражнение 43. Выделите класс невегетарианских пицц *NonVegetarianPizza* (выражение вида $\neg$).

№2. Разработка системы персонализации на основе совместной фильтрации

Целью практической работы является приобретение навыков интеграции системы веб-персонализации на основе совместной фильтрации, при пользователи получают рекомендации на основе степени схожести с другими пользователями (профили) и на основе степени схожести продуктов между собой.

Vogoo PHP Lib – бесплатная PHP-библиотека, позволяющая вебмастеру добавить возможности персонализации на основе совместной фильтрации, а именно – выдавать рекомендации пользователям на основе их вкусов и предпочтений и показывать полезную информацию на основе их схожести с другими пользователями.

В начале работы необходимо подключить библиотеку:

```
include('vogoo.php');
```

Вызов функций библиотеки осуществляется следующим образом:

```
vogoo->название_функции(параметр 1,параметр2);
```

Основными параметрами на входе являются идентификаторы пользователей, идентификаторы продуктов и оценки в диапазоне от 0.00 до 1.00 (числа с плавающей запятой).

Изучите работу основных функций библиотеки:

1. Откройте в браузере ссылку на главную страницу index.php Выберите свою фамилию в выпадающем списке и оцените фильмы по шкале от 0,5 до 5 звезд. Если вы не видели фильм или не уверены в оценке, выберите вариант «Not seen» – только по таким фильмам вы сможете получить персональные рекомендации. Нажмите Submit. В бригаде рейтинг дает каждый студент.

Функция $vogoo->set_rating($member_id,$product_id,$rating) присваивает продукту $product_id оценку, выставленную пользователем $member_id. Значение $rating должно быть в диапазоне [0..1].

2. Перейдите по ссылке General Information. На этой странице представлена сводная информация обо всех пользователях и продуктах.

Функция $vogoo->member_num_ratings($member_id) возвращает число продуктов, получивших оценку пользователя $member_id.

Функция $vogoo->member_average_rating($member_id) возвращает среднее арифметическое оценок, выставленных пользователем $member_id.

Функция $vogoo->product_num_ratings($product_id) возвращает число оценок, полученных продуктом $product_id.

Функция $vogoo->product_average_rating($product_id) возвращает среднее арифметическое оценок, полученных продуктом $product_id.

3. Выберите себя в списке пользователей. На странице представлена Карточка пользователя.

Функция $vogoo->member_ratings($member_id) возвращает массив продуктов, имеющих оценку пользователя $member_id. Каждый ряд массива содержит три элемента: идентификатор продукта, оценку в диапазоне от 0 до 1 и время, когда была выставлена оценка.

Функция $vogoo->get_rating($member_id,$product_id) возвращает массив, первым элементом которого является рейтинг, который пользователь $member_id выставил продукту $product_id. Второй элемент – время, когда была выставлена оценка.

Функция $vogoo->member_similarity($member_id1, $member_id2, $cat = 1) возвращает значение степени близости (целое число в диапазоне от 0 до 100) между пользователями $member_id1 и $member_id2 в категории $cat (для фильмов $cat = 1).

Функция $vogoo->member_k_similarities($member_id,$k, &$similarities,$cat = 1) вычисляет $k максимальных степеней близости для пользователя $member_id в категории $cat, и сохраняет их значение в переменную $similarities. Каждый ряд массива содержит два элемента: идентификатор пользователя и степень близости с ним. Это очень важная функция, поскольку массив $similarities служит аргументом для других функций, вычисляющий рекомендации.

4. Для получения списка рекомендаций перейдите по ссылке Click here to see the recommendations for.

Функция $vogoo->member_k_recommendations($member_id, $k,&$similarities, &$recommendations,$cat = 1,$filter = false) записывает в переменную $recommendations массив, содержащий до $k рекомендаций для пользователя $member_id. Аргумент $similarities – это массив, предварительно заполненный функцией $vogoo->member_k_similarities (см. п.3). Элементами массива $recommendations являются: идентификатор пользователя, степень его близости к пользователю $member_id, идентификатор продукта, оценка, полученная этим продуктом, и время, когда была выставлена оценка.

Функция $vogoo->get_product_recommendation($product_id, &$similarities,$cat = 1) возвращает оценку продукта $product_id, выставленную пользователем с максимальной степенью близости к текущему пользователю. Элементами массива являются: идентификатор пользователя, степень его близости к текущему пользователю, его оценка продукта и время, когда оценка была выставлена.

5. Вернитесь на страницу General Information и выберите любой фильм из списка. На странице представлена Карточка фильма.

Функция $vogoo->product_ratings($product_id) работает по аналогии с функцией $vogoo->member_ratings($member_id), возвращая массив пользователей, выставивших оценку продукту $product_id.

Функция $vogoo->get_product_ratings_by_similarity ($product_id,&$similarities) возвращает массив оценок, выставленных продукту $product_id и отсортированных в порядке уменьшения степени близости.

Выполните задания:

1. В разделе Single Rating в Карточке пользователя вывести оценку, которую текущий пользователь выставил фильму «Матрица».

2. В разделе Single Similarity вывести степень близости со студентом, работающим с вами в одной бригаде.

3. В разделе Similarity List вывести список из 5 студентов с наибольшей степенью близости.

4. Организовать раздел Linked products, используя функцию $vogoo_items->get_slope_items.

Функция $vogoo_items->get_slope_items($product_id, $min_nr_links = 2,$cat = 1,$filter = false) возвращает массив продуктов, *связанных* с продуктом $product_id по алгоритму Slope. Каждая строка массива состоит из двух элементов – идентификатора продукта и разницы рейтингов продукта $member_id и продукта в массиве. Отсортировано по убыванию. Для корректной работы этой функции значение $min_nr_links (связей между продуктами) должно быть >= 2**.**

Пример кода, выводящего первую строку массива:

```
<?php
$slope_items = $vogoo_items->get_slope_items($pdt_id,$min_nr_links = 2,$cat =
1,$filter = false);
echo $slope_items[0][0];
echo " ";
echo $slope_items[0][1];
?>
```

Раздел реализовать в виде таблицы с колонками Название продукта и Разница рейтингов.

5. В Карточке фильма организовать раздел Not rated yet, используя функцию $vogoo_items->member_predict_all.

Функция $vogoo_items->member_predict_all($member_id,$cat = 1,$filter = false) возвращает прогноз оценок, которые пользователь $member_id даст еще не оцененным продуктам.

Пример кода, выводящего первую строку массива:

```
<?php
$member_predicts = $vogoo_items->member_predict_all($member_id = 1,$cat =
1,$filter = false);
echo $member_predicts[0][0];
echo " ";
echo $member_predicts[0][1];
?>
```

Раздел реализовать в виде таблицы с колонками Название продукта, Прогноз рейтинга.

6. Произвести модификацию раздела «Рекомендации на основе алгоритма Slope», который будет включать только те фильмы, для которых вышли сиквелы.

7. Исключить из рассмотрения несколько фильмов, в которых студент изначально не заинтересован и по какой-то причине не хочет видеть их в рекомендации, даже если они соответствуют его интересам.

8. Произвести модификацию раздела «Прогноз рейтинга», который будет включать только комедии и мультфильмы.

9. По аналогии с Amazon.com организовать на Карточке фильма иллюстрированный раздел «Пользователи, которым понравился этот фильм, также предпочли...».

Отчетом по практической работе служат файлы demomember.php, demoproduct.php и пояснительная записка в электронном виде.

Контрольные вопросы

1. *Охарактеризуйте предмет интеллектуальных интернет-технологий.*
2. *Какие требования предъявляются к «Мудрой» Всемирной Паутине нового поколения?*
3. *Как вы понимаете категорию «онтология» в контексте интеллектуальных интернет-технологий?*
4. *Каким образом онтология трансформируется в простой словарь, пассивный словарь, активный словарь, таксономию?*
5. *Охарактеризуйте комплекс рекомендаций W3C, связанных с Семантической Всемирной Паутиной.*
6. *Какие инструменты инженерии онтологий существуют?*
7. *Как вы понимаете категорию «искусственный агент»?*
8. *Дайте свою классификацию искусственных агентов.*
9. *В чем особенность описания взаимодействия между агентами при помощи математического аппарата теории отношений?*
10. *Охарактеризуйте реактивную, делиберативную и гибридную архитектуры MAC.*
11. *Раскройте сущность мобильных агентов.*
12. *В чем особенность «индивидуального маркетинга» в контексте интернет-приложений?*
13. *Как вы понимаете основные функции персонализации?*
14. *Раскройте сущность адаптивной гипермедиа.*
15. *Что такое «пользовательский профиль»?*
16. *Дайте формальную постановку задачи персонализации.*
17. *Предложите альтернативные оценки эффективности рекомендаций на основе методов информационного поиска.*
18. *Что такое ассоциативные правила?*
19. *Опишите архитектуру типовой системы персонализации.*
20. *Какие виды поисковых алгоритмов существуют?*

21. *Охарактеризуйте способы ранжирования результатов поиска в различных моделях.*
22. *Раскройте сущность теоретико-графовых подходов к семантическому поиску.*
23. *С какими проблемами сталкиваются разработчики поисковых систем в интернете?*
24. *В чем основные отличия грид-вычислений от технологий peer-to-peer и кластерных вычислений?*
25. *Какие вы видите перспективы развития грид-технологий?*
26. *Раскройте сущность теории малого мира.*
27. *Охарактеризуйте методы совместной фильтрации.*
28. *Дайте свой прогноз развития интеллектуальных интернет-технологий.*