ПОСТРОЕНИЕ ИЕРАРХИИ КЛАССОВ ПО ТЕКСТОВЫМ ОПИСАНИЯМ

А.А. Бреслав, А.П. Лукьянова, М.А. Коротков
Научный руководитель – к.ф.-м.н., с.н.с. Ф.А. Новиков
(Санкт-Петербургский государственный политехнический университет)

Цель настоящей работы – автоматизация объектно-ориентированного анализа предметной области, описанной на естественном (английском) языке. Описываемый подход позволяет строить первичную иерархию классов, в дальнейшем уточняя ее посредством автоматического рефакторинга и статического анализа текста. Распознаются классы, наследование, атрибуты и методы.

Введение

Анализ – один из важнейших этапов жизненного цикла программной системы, когда, изучая предметную область, разработчики выделяют абстракции, определяющие концептуальную структуру будущей системы. Существует не так уж много различных методик анализа. Гради Буч (Grady Booch) в работе [1] выделяет три классических подхода (классическая категоризация, концептуальная кластеризация, теория прототипов) поведенческий подход, основанный на ответственности, и использование уже существующих достижений (паттерны). Кроме того, Буч упоминает работу Рассела Дж. Аббота (Russel J. Abbott) [2], в которой описан неформальный метод анализа, основанный на текстовом описании. Принцип, по которому Аббот предлагал выделять классы и объекты, основывается на синтаксической роли слов в предложении: подлежащее, скорее всего, является объектом, сказуемое описывает его поведение, определения описывают атрибуты и т.д.

С момента публикации Абботом этой работы метод был уточнен и расширен. Аббот опирался в основном на части речи, считая, например, что определение – это непременно прилагательное. Позднее было отмечено, что не только прилагательные, но и существительные могут играть эту роль [4, 5]. Были предложены специальные трактовки для отдельных слов и конструкций-связок (таких как «all», «any», «within», «can be» и т.д.) [6–8].

Были созданы и автоматические средства анализа. Некоторые из них распознают лишь объекты, не выделяя классов [9, 10]. Но есть и такие, которые распознают классы и ассоциации между ними [11, 12]. Однако ни одна из известных нам систем не обнаруживает наследования.

Существуют системы, использующие доменную информацию в качестве исходных данных (в дополнение к тексту) [13, 14], но эта информация слишком конкретна: пользователю фактически предлагается самостоятельно определить объекты и методы.

Выходные данные подобных систем могут быть формализованы по-разному. Весьма распространено использование для этой цели различных диаграмм. Системы, описанные в работах [13, 14], могут строить диаграммы UML [15].

Целью настоящей работы является построение системы, позволяющей обнаруживать классы, их атрибуты и методы, а также выделять отношение наследования.

Постановка задачи

В качестве примера рассмотрим следующий текст, описывающий предметную область «Магазин».

The shop sells food. The shop sells drinks. The food's price is a double. The drinks' price is a double. There is a shop assistant in the shop. The consultant, shop assistant, consults a customer. When the customer comes the customer buys food. The customer buys drinks. There is a cash desk in the shop. The cash desk gives out a cheque.

Этот текст описывает взаимодействие следующих сущностей (классов): магазин (shop), продавец (shop assistant), консультант (consultant), еда (food), напитки (drinks), касса (cash desk), чек (cheque) и покупатель (customer). Сущности имеют атрибуты: для еды и напитков – цена (price), для магазина – продавец и касса. Также они имеют методы: магазин продает (sells), консультант консультирует (consults), покупатель приходит (comes) и покупает (buys), касса выдает чек (gives out a cheque).

Таким образом, текст фактически описывает структуру классов. Задача предлагаемого метода – распознать эту структуру и сформировать классы, связанные отношениями наследования.

Математическая модель

В качестве модели для работы с текстом был выбран граф объектных отношений. Этот граф отображает информацию о семантической структуре текста, он представляет собой неоднородную бинарную семантическую сеть [16].

Граф строится из подграфов, каждый из которых представляет одно предложение исходного текста. Принцип построения графа основывается на методе Аббота, т.е. считается, что подлежащее, как правило, представляет объект, сказуемое – метод, зависимые слова описывают главное, т.е. являются его атрибутами или параметрами, и т.д.

Каждое предложение порождает некоторый подграф (см. рис. 1). Вершины в этом подграфе представляют роли тех или иных слов или понятий, а ребра – связи между вершинами, выражаемые бинарными отношениями.

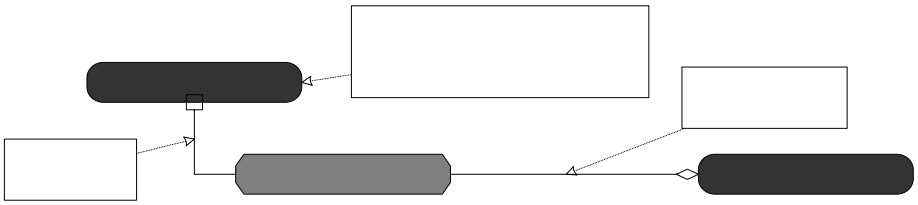

Рис. 1. Граф предложения «There is a shop assistant in the shop»
(«В магазине есть продавец»)

Одна вершина может соответствовать словосочетанию, и одно слово может порождать несколько вершин. Последняя ситуация реализуется, например, при обнаружении атрибута ранее неизвестного типа.

В предложении на рис. 1 словосочетание «shop assistant» трактуется как одно понятие, а это понятие представляется двумя вершинами в графе: объектом, играющим роль типа, и атрибутом другого объекта.

В модели используются следующие типы вершин (см. рис. 2).

- Объект – упоминание объекта, т.е. некоторой описываемой единицы.
- Атрибут – упоминание характеристики, т.е. описание аспекта состояния.
- Метод – упоминание действия некоторого объекта, т.е. описание аспекта поведения.
- Параметр – упоминание параметра (характеристики) действия.

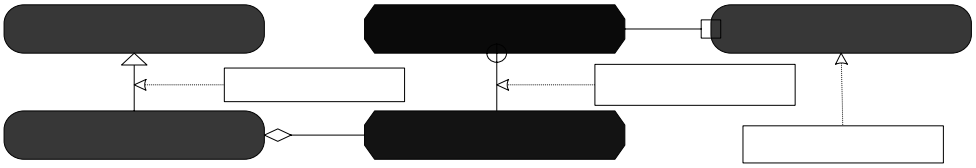

Рис. 2. Граф предложения «The consultant, shop assistant, consults a customer»
(«Консультант, будучи продавцом, консультирует покупателя»)

Используются следующие типы ребер (см. рис. 2).

- Принадлежность элемента интерфейса конкретному объекту.

- Наследование одним объектом свойств другого. Если объект А наследуется от объекта В, значит, класс, к которому принадлежит А, является частным случаем класса, к которому принадлежит В.
- Использование одним объектом свойств другого. Если такое ребро указывает на объект, это означает использование объекта в целом. Также оно может указывать на конкретный атрибут или метод. Началом ребра может быть не только объект, но и один из его методов.
- Использование методом данного параметра.
- Возвращаемое значение метода.

Все вершины и связи локализованы в одном предложении, поскольку именно предложение является наиболее крупным подразделением текста, в семантической целостности которого можно быть уверенным. Графы предложений могут быть связаны между собой, например, если установлено, что местоимение в одном предложении представляет некоторое полнозначное слово в другом[1].

Формирование объектной модели ведется по следующему общему принципу (см. рис. 3): объекты с одинаковыми именами, обнаруженные в различных предложениях объединяются в классы. Интерфейсы этих классов строятся как объединение интерфейсов всех объектов, входящих в класс, т.е. все атрибуты и методы, связанные с этими объектами отношением принадлежности, составляют интерфейс нового класса.

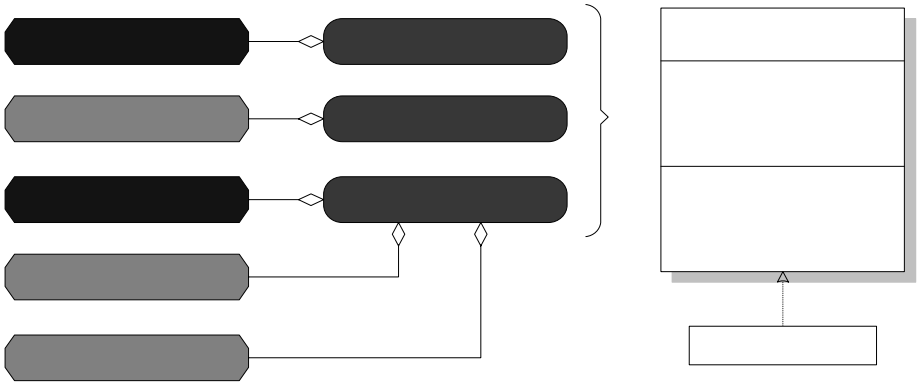

Рис. 3. Объединение объектов в класс

Терминология

Далее под словами «группа атрибутов» (объектов, методов) будем понимать все вершины соответствующего типа, имеющие одинаковые имена (эти вершины могут находиться в разных предложениях).

Вершины графа объектных отношений являются промежуточным звеном между словами в тексте и элементами конечной структуры (классами, атрибутами и методами). Группа вершин одного типа порождает класс, атрибут или метод.

Заметим, что объекты не являются элементами конечной структуры (они объединяются в классы), поэтому понятие «объект» обозначает именно вершину графа, т.е. экземпляр класса. Группа объектов формирует класс (см. рис. 4).

[1] В настоящее время задача об установлении таких связей не решена

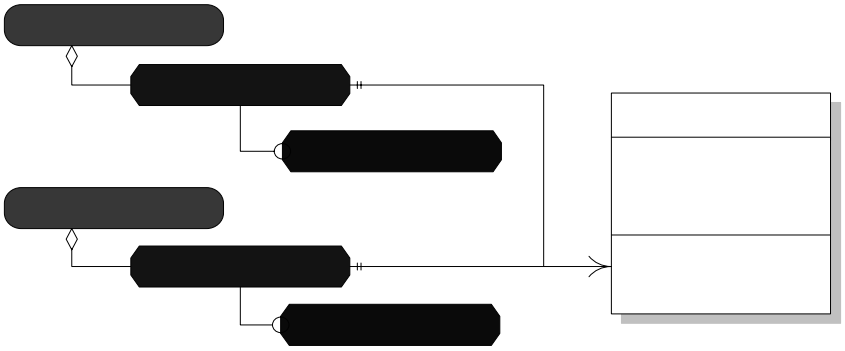

Рис. 4. Порождение метода группой вершин

Дополнительные структуры данных

Для дальнейшей работы потребуются дополнительные структуры данных. Все слова и словосочетания, представленные вершинами графа, заносятся в *словарь*, причем для каждого слова хранится список вершин каждого типа, порожденных этим словом.

Также потребуется понятие *категории*. Категория – это группа слов, объединенных общим смыслом. Структура категорий может быть очень сложной, в идеале она напоминает файловую систему, где роль файла играет слово из словаря, а роль каталога - категория. Как и в файловой системе, каждая категория имеет уникальное имя.

В тексте могут встретиться термины, специфичные для предметной области, которых нет в предопределенном словаре категорий. Здесь возникает задача самообучения программы. В прототипе эта проблема решена самым простым способом: каждое неизвестное слово автоматически помещается в категорию с таким же именем.

Сформированные элементы конченой структуры помещаются в специальное *хранилище*, из которого можно сформировать объявления классов на любом языке программирования или диаграмму UML.

Спецификаторы классов

Одним из важнейших в нашей модели является понятие *спецификатора*.

Спецификатор класса – это атрибут, достаточно выразительный для того, чтобы его наличие говорило о принадлежности объекта отдельному классу.

Расширяя пример с магазином, опишем некоторую классификацию товаров.

*The shop sets **discounted price**. The **discounted price** is lower than **usual price**.*

Мы описали классы «Цена со скидкой» и «Обычная цена» (см. рис. 6), поскольку для объектов этих классов значение вычисляется по-разному. Фактическим критерием для нас служит разница в способе вычисления, но анализ принципа работы метода (вычисления стоимости) по неформальному описанию – задача крайне трудная, и тут на помощь приходит эвристический механизм выделения спецификаторов.

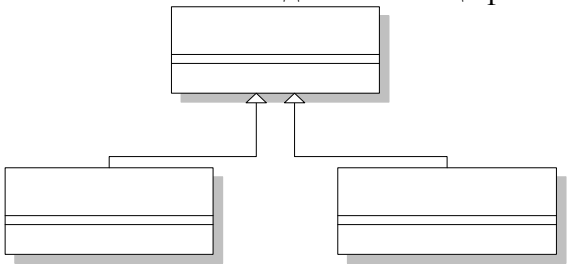

Рис. 5. Спецификаторы классов

Дело в том, что, описывая некоторый специфичный аспект поведения класса, мы вынуждены каждый раз указывать спецификатор, чтобы отличать разновидности товаров. Так, в приведенном примере использованы слова «discounted price» и «usual price». Вершины «discounted» и «usual» являются атрибутами, однако фактически они неразрывно связаны с определяемым понятием и обозначают *подклассы*.

Такое положение вещей позволяет нам использовать статистические данные о встречаемости атрибута для принятия решения о выделении спецификатора. Выделение спецификаторов очень важно, поскольку оно позволяет установить связи наследования между классами.

Для выделения спецификаторов классов рассмотрим те слова из словаря, которые являются именами объектов. Если ни один из таких объектов не имеет входящих ребер наследования, рассмотрим все атрибуты, принадлежащие этим объектам.

Не любой атрибут может служить спецификатором. Некоторые атрибуты могут изменять свои значения в процессе жизненного цикла объекта, но спецификатор является элементом индивидуальности объекта и изменяться не может, поэтому, если есть явное указание на использование значения атрибута некоторым методом или объектом (отношение использования), то выделение спецификатора по этому атрибуту невозможно. Поэтому атрибуты, связанные отношением использования, далее не рассматриваются.

Для каждого класса (группы объектов с одинаковыми именами) рассмотрим все категории, в которые входят имена выбранных атрибутов. Выберем категорию, элементы которой наиболее часто встречаются с выбранными объектами, и на каждый элемент в среднем приходится достаточно большое число упоминаний. Такая категория будет хорошим кандидатом на роль спецификатора (рис. 7).

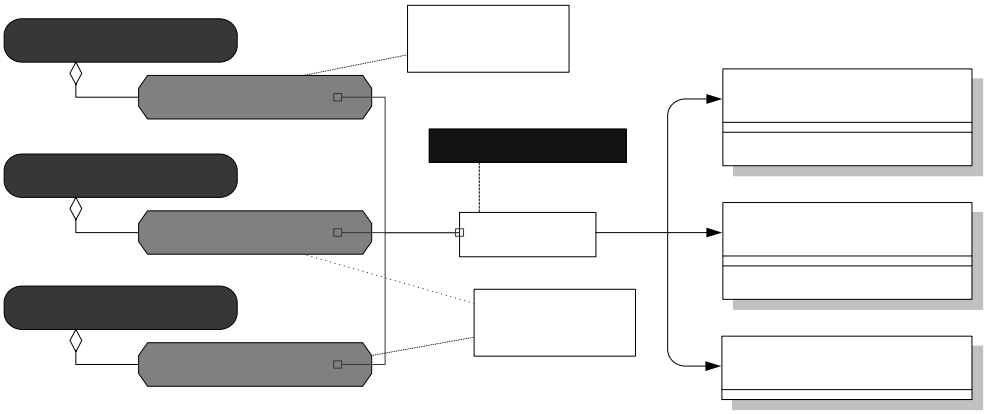

Рис. 6. Замена значений атрибутом, по спецификаторам

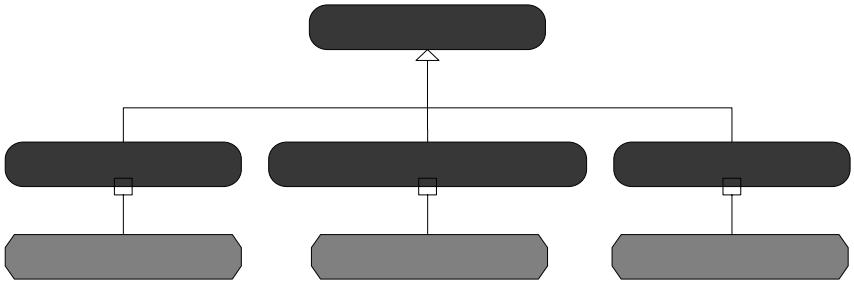

Рис. 7. Выбор суперкласса для типов атрибутов

Почему в качестве спецификатора выбирается категория? Дело в том, что фактическое имя атрибута (например, «цвет») может в тексте явно не указываться, встречаться могут только его значения (например, «желтый», «зеленый», «синий»). Эти значения должны быть объединены в единый атрибут с именем категории, но делать это до выделения спецификаторов нельзя. Таким образом, вершина типа «атрибут» может представлять собой не собственно атрибут класса, а лишь его значение, а если мы выделяем

специфицированный класс по одному из значений данного атрибута, то нужно выделить и другие классы – по остальным значениям.

Теперь добавим в хранилище новый класс, который будет родительским для выделенных по спецификатору. Каждый элемент из выбранной категории порождает новый класс, наследуемый от этого.

Типизация атрибутов

После выделения спецификаторов классов у объектов одного класса (т.е. с одним именем) могут быть атрибуты, имеющие одинаковые имена, но разные типы (в одном предложении использовался один спецификатор класса, а в другом другой). Для решения этой проблемы предлагается выбрать суперкласс для типов этих атрибутов и установить его как общий тип (рис. 8).

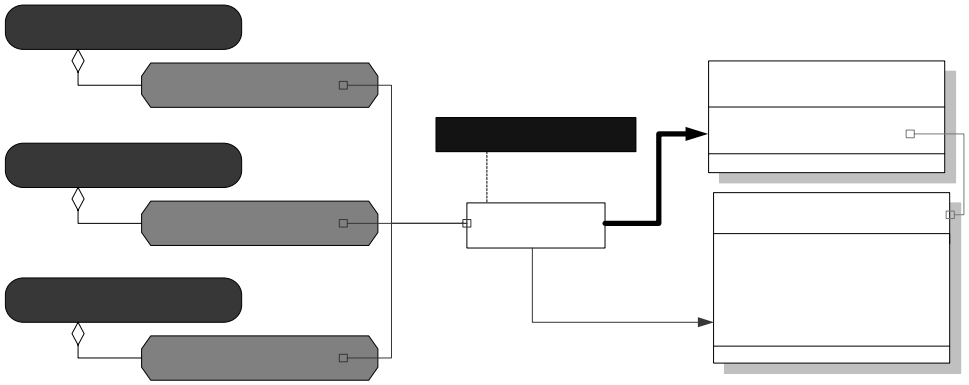

Рис. 8. Замена значений атрибутом

Для этого будем спускаться сверху вниз по иерархии наследования, будем рассматривать все атрибуты каждого класса. Для каждого атрибута возможны три принципиально разные ситуации:

- для атрибута известен тип;
- тип неизвестен, но известна категория;
- не известны ни тип, ни категория (этот случай реализуется для специфицированных классов, имена которых не принадлежат категориям).

По этому признаку атрибуты разбиваются на три списка.

Теперь необходимо обратиться к потомкам текущего класса и рассмотреть их атрибуты. Встретив атрибут, имя которого уже содержится в первом списке, добавим его в этот список. Если категория рассматриваемого атрибута встретилась во втором списке, добавить его во второй список.

После завершения обхода поддерева иерархии наследования все атрибуты родительского класса и часть атрибутов потоков будут распределены по трем спискам. Теперь нужно перенести эти атрибуты в хранилище.

Первый список – типизированные атрибуты. В этом списке могут содержаться атрибуты с одним именем, но разными типами. Для таких атрибутов необходимо найти общий суперкласс их типов и установить его как общий тип для всех. Атрибут удаляется из интерфейсов классов-наследников и сохраняется только в родительском классе, поскольку потомки унаследуют его автоматически.

Второй список – категории. Имена категорий из второго списка добавляются в хранилище как атрибуты. Областями их значений будут константы с именами конкретных слов из категории. Например, категория «цвет» порождает атрибут Цвет со значениями (красный, желтый, зеленый) (рис. 8).

Спецификаторы методов

Спецификатор метода – это параметр, достаточно выразительный для того, чтобы его наличие говорило о наличии отдельного метода.

Принцип работы алгоритма спецификации методов во многом схож с алгоритмом спецификации классов. Параметры играют роль атрибутов, а вместо наследования мы просто сохраняем неспецифицированный метод, чтобы специфицированные могли его использовать. Та же схема – с типизацией параметров, но здесь богатую почву для исследований открывает возможность перегрузки. В настоящее время перегрузка не поддерживается, но ее наличие даст возможность очень гибко работать с методами.

Дополнительно необходимо обрабатывать типы возвращаемых значений функций.

Наследование

Формирование связей наследования по тексту (отношение наследования) и по спецификаторам является довольно точной, но не очень обширной эвристикой. Рассматриваемый далее метод позволяет в значительной мере использовать данные, хранимые структурой категорий для выявления отношения наследования в предметной области.

К настоящему моменту работа с графом отношений завершена. Все выделенные элементы конечной структуры находятся в хранилище.

Наследование по категориям

Для каждой категории рассмотрим множество классов из хранилища, названных словами этой категории. Эти классы могут быть родственными, поскольку их объединяет семантика названий (рис. 9). Если в интерфейсах этих классов имеются схожие элементы, имеет смысл создать для них общий суперкласс, и вынести туда общие атрибуты и методы. При подъеме атрибутов (выбранных по общности имен) необходимо производить типизацию по методу, изложенному выше, т.е. выбирать в качестве типа общий суперкласс типов имеющихся объектов. Методы также выбираются по общности имени и приводятся к единой сигнатуре (перегрузка не поддерживается) по тому же принципу, по какому классы строятся из объектов: производится типизация параметров.

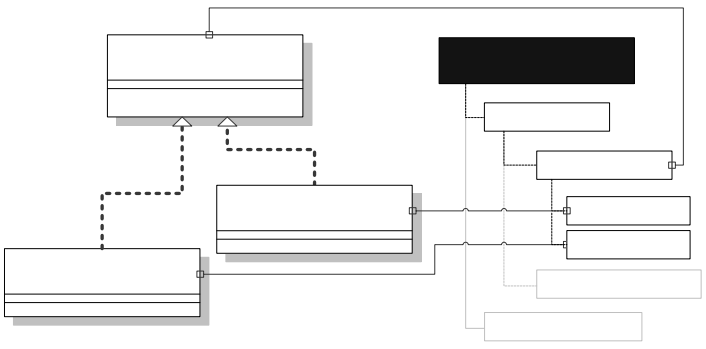

Рис. 9. Перенос ребер из категорий в хранилище

Улучшение имеющейся иерархии наследования

В процессе формирования иерархии наследования могли возникнуть ситуации, требующие применения рефакторинга. Рассмотрим следующий пример: интерфейсы некоторых (но не всех) подклассов класса **Sample** имеют непустое пересечение, т.е.

возникла необходимость в подъеме поля или подъеме метода (см. [17], рис. 11). Для улучшения имеющейся ситуации будем рассматривать классы, поднимаясь по иерархии наследования снизу вверх. Для каждого класса рассмотрим множества его непосредственных потомков и, находя общие атрибуты и методы, будем выделять новые классы, наследуя их от текущего.

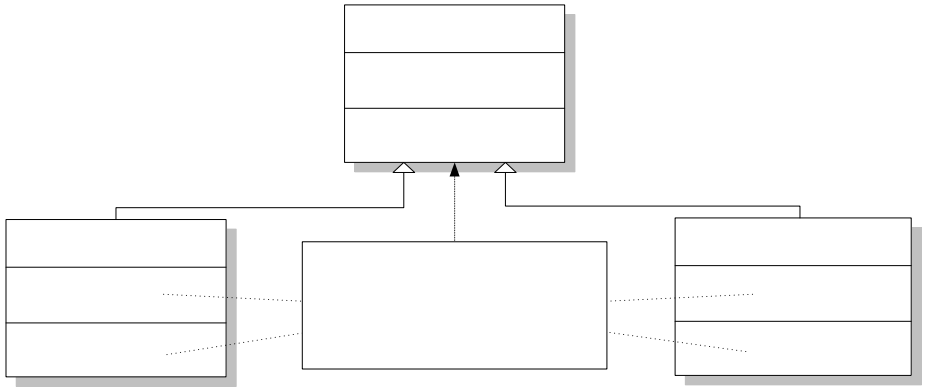

Рис. 10. Рефакторинг: подъем поля и метода

Реализация описанного метода

Разработанный прототип системы, названный ObjectSage, позволяет пользователю ввести текстовое описание и строит по нему граф объектных отношений, диаграмму классов и заголовочный файл C++ (см. рис. 11).

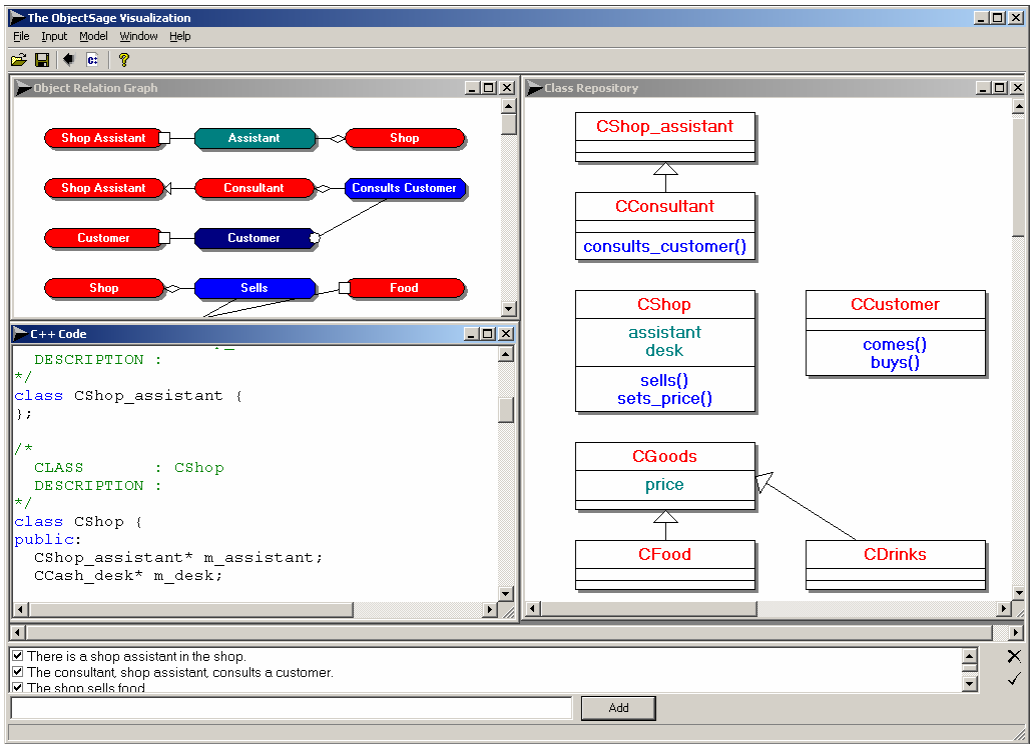

Рис. 11. Скриншот системы ObjectSage

Анализ английского языка и построение графа объектных отношений выполняется с помощью *Link Grammar Parser* [18] – разработки Carnegie Mellon University. С помощью этого пакета удалось выявить четыре из пяти типов ребер (кроме возвращаемого значения метода) и все типы вершин. Задачу о связях между предложениями по местоимениям средствами *LGP* решить не удалось. Разработанный прототип способен обрабатывать простые предложения без местоимений, косвенной речи, страдательного

залога и прочих сложных конструкций. Поддерживается работа с категориями, выделение спецификаторов и рефакторинг конечной структуры (подъем общих элементов в суперкласс). Распознаются перечислимые и примитивные типы.

Используемая структура категорий – плоская, т.е. нет вложенных категорий. Считается, что каждое слово принадлежит ровно одной категории.

Заключение

Работа с прототипом системы показывает, что предлагаемые методы пригодны для решения поставленной задачи. Применяя их, удается получить качественные модели предметной области (принимая во внимание ограничения, накладываемые средствами анализа текста). Вмешательство пользователя в процесс работы не требуется. Вся доменно-специфичная информация организуется в виде структуры категорий, причем не требуется никакого семантического описания понятий – только классификация.

Получаемые модели достаточно информативны для генерации по ним декларативной части программы, что говорит о том, что предлагаемые методы могут быть использованы при решении задач генеративного программирования.

Для достижения более полного объема функциональности необходимо использовать более сложные методы анализа текста, причем нет необходимости качественно изменять алгоритмы, предлагаемые в данной работе – необходимо лишь реализовать те концепции, которые еще не реализованы.

Литература

1. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++ // СПб.: Невский диалект, 1999.
2. Abbott R.J. Program Design by Informal English Descriptions // Communications of the ACM. 1983. Vol. 26, 11. P. 882–894.
3. Buchholz H., Dusterhoft A., Thalheim B. Capturing Information on Behaviour with the RADD_NLI: A Linguistic and Knowledge Base Approach // Proceedings of Second Workshop Application of Natural Language to Information System, IOS Press, Amsterdam, 1996., P. 185–196.
4. Kristen G. Object Orientation: The KISS-Method: From Information Architecture to Information Systems // Addison-Wesley, Reading, Mass., 1994.
5. Rolland C., Proix C. A Natural Language Approach for Requirements Engineering // Proceeding on Conference Advanced Information Systems Engineering. Springer-Verlag, Manchester, UK. 1992. P. 257–277.
6. Tjoa A.M., Berger L. Transformation of Requirements Specification Expressed in Natural Language into an EER Model// Proceeding: 12th International Conference on ER Approach. Springer-Verlag, Berlin. 1993. P. 206–217.
7. Juristo N., Moreno A.M. How to Use Linguistic Instruments for Object-Oriented Analysis // IEEE Software, May/June 2000. P. 80–89.
8. Mich L. From Natural Language to Object Oriented Using the Natural Language Processing System LOLITA // Natural Language Engineering, 2(2), 1996. P.161–187.
9. Delisle S., Barker K., Biskri I. Object-Oriented Analysis: Getting Help from Robust Computational Linguistic Tools. // The Fourth International Conference on Applications of Natural Language to Information Systems (OCG Schriftenreihe 129), Klagenfurt, Austria, 1999, 167–171.
10. Harmain H.M., R. Gaizauskas. CM-Builder: An Automated NL-based CASE Tool // In Proceedings of the 15th IEEE International Conference on Automated Software Engineering (ASE'2000), 2000. P. 45–53.

11. Harmain H.M., Gaizauskas R. CM-Builder: A Natural Language-based CASE Tool // Journal of Automated Software Engineering, 10. 2003. P. 157–181.
12. Börstler J. User-Centered Requirements Engineering in RECORD – An Overview // The Nordic Workshop on Programming Environment Research, Proceedings NWPER'96, Aalborg, Denmark, May 1996. P. 149–156.
13. Overmyer S.L.V., Rambow O. Conceptual Modeling through Linguistics Analysis Using LIDA // 23rd international conference on Software engineering. July 2001.
14. Perez-Gonzalez H.G., Kalita J.K. GOOAL: A Graphic Object Oriented Analysis Laboratory // OOPSLA'02, November 2002. P. 38–39.
15. Фаулер М., Скотт К. UML. Основы. // СПб: Символ-Плюс. 2003. 192 с.
16. Sowa J.F. Semantic Networks // www.jfsowa.com/pubs/semnet.htm
17. Фаулер М. Рефакторинг: улучшение существующего кода. // СПб: Символ-Плюс. 2003. 432 с.
18. Temperley D., Sleator D., Lafferty J. Link Grammar Parser // Carnegie Mellon University, http://www.link.cs.cmu.edu/link/