OSTIS-2013
(Open Semantic Technologies for Intelligent Systems)

УДК 004.822:514

СЕМАНТИЧЕСКИЕ ЯЗЫКИ ДЛЯ ОПИСАНИЯ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММ, ОРИЕНТИРОВАННЫХ НА ОБРАБОТКУ ЗНАНИЙ

Пивоварчик О.В.

[*] *Белорусский государственный университет информатики и радиоэлектроники, г. Минск, Республика Беларусь*

pivovarchyk@tut.by

В работе описаны семантические языки представления знаний, используемые для создания баз знаний интеллектуальных help-систем для разработчиков программ, ориентированных на обработку знаний. Семантические языки являются sc-языками и определяются своим расширением множества ключевых узлов sc-кода.

Ключевые слова: интеллектуальная help-система, технология проектирования программ, язык программирования, язык представления знаний, база знаний.

Введение

Интеллектуальная help-система для разработчиков программ, ориентированных на обработку знаний, представляет собой документацию по соответствующей технологии проектирования программного обеспечения в виде интеллектуальной справочной и обучающей системы [Голенков и др., 2012]. Интеллектуальная help-система обеспечивает всестороннее информационное обслуживание пользователей, обучение пользователей, что приводит к увеличению количества разработчиков интеллектуальных систем и сокращению сроков проектирования интеллектуальных систем.

Для проектирования help-системы используется открытая семантическая технология компонентного проектирования интеллектуальных систем (Open Semantic Technologies for Intelligent System, OSTIS). В основе OSTIS лежит семантическая модель представления знаний, которая использует семантическую сеть с базовой теоретико-множественной интерпретацией и представление знаний в sc-коде. OSTIS предполагает этапы проектирования интеллектуальной системы: проектирование базы знаний, проектирование машины обработки знаний, проектирование пользовательского интерфейса.

Для представления информации в базе знаний help-системы используется универсальный открытый язык представления знаний sc-код. SC-код базируется на аппарате однородных семантических сетей и включает в качестве подъязыков специализированные семантические языки представления и обработки знаний различных предметных областей [Электронный ресурс], [Голенков и др., 2001а]. Подъязыки sc-кода (далее – sc-языки) разрабатываются путем выделения необходимого набора ключевых узлов. В работе предлагаются sc-языки для представления знаний технологии проектирования программ, ориентированных на обработку знаний.

Технология разработки программ, ориентированных на обработку знаний, включает следующие компоненты:

- теория программ: язык программирования, язык представления знаний;
- библиотека типовых многократно используемых компонентов (ip-компонентов);
- инструментальное средство проектирования программ;
- методика проектирования программ;
- методика обучения проектированию программ.

Следовательно, основой построения базы знаний интеллектуальной help-системы являются формальные модели соответствующих компонентов: формальная модель языка программирования, формальная модель языка представления знаний, формальная модель инструментального средства, формальная модель библиотеки компонентов, формальная модель методики проектирования программ, формальная модель методики обучения программированию. А также, иерархическая структура разделов, каждый из которых описывает некоторый элемент технологии. Разделы базы

знаний именуются, нумеруются и являются оглавлением.

В соответствии с компонентами технологии выделим следующие sc-языки:

- семантический язык для описания языка программирования и языка представления знаний $L_{program}$;
- семантический язык для описания интегрированной среды разработки программ L_{ide};
- семантический язык для описания методики проектирования программ L_{method};
- семантический язык для описания методики обучения программированию $L_{training}$.

1. Семантический язык для описания языка программирования и языка представления знаний

Описание формальной модели языка программирования можно задать его спецификацией. Спецификация содержит описание синтаксиса и семантики языка.

Синтаксис определяет множество допустимых конструкций. Существует два способа описания синтаксиса: абстрактный синтаксис и конкретный синтаксис. Абстрактный синтаксис представляет собой абстрактную модель синтаксических конструкций языка, включающую правила построения корректных программных текстов, не зависящую от конкретного вычислителя. Конкретный синтаксис задает запись конструкций языка в конкретной нотации по заданной абстрактной модели. Основными формальными методами определения синтаксиса языка программирования являются форма Бэкуса-Наура (далее – БНФ) и контекстно-свободные грамматики, расширенная форма Бэкуса-Наура, синтаксические графы [Себеста, 2001], [Пратт и др., 2002]. В данной работе за основу взят метод БНФ. Предлагаемый в работе язык позволяет представить в виде семантической сети информацию о правилах БНФ.

Семантика разъясняет смысл синтаксических конструкций. Наиболее распространенными методами описания семантики языков программирования являются: денотационный, операционный, аксиоматический, алгебраический [Вольфенгаген, 2001], [Mitchell, 1996], [Shmidt, 1986]. В данной работе для описания языков программирования будут использоваться денотационный и операционный методы.

Семантический язык $L_{program}$, используемый для описания языков программирования, ориентированных на обработку знаний, задается объединением множеств

$$L_{program} = C_{syntax} \cup S_{syntax} \cup R_{syntax} \cup C_{semantic} \cup S_{semantic} \cup R_{semantic},$$

где C_{syntax} – множество ключевых узлов, предназначенных для описания синтаксиса языка программирования; S_{syntax} – множество утверждений, предназначенных для описания синтаксиса языка программирования; R_{syntax} – множество отношений, определенных на множестве ключевых узлов C_{syntax} и S_{syntax}; $C_{semantics}$ – множество ключевых узлов, предназначенных для описания денотационной и операционной семантик языка программирования; $S_{semantics}$ – множество утверждений, предназначенных для описания семантики языка программирования; $R_{semantics}$ – множество отношений, определенных на множестве ключевых узлов $C_{semantics}$ и $S_{semantics}$.

1.1. Описание синтаксиса

Для описания абстрактного синтаксиса C_{syntax} включает множество ключевых узлов, обозначающих типы синтаксических конструкций (синтаксических доменов), S_{syntax} – множество утверждений, описывающих правила построения синтаксических конструкций. Для определения конкретного синтаксиса множество C_{syntax} дополняется ключевыми узлами, необходимыми для задания определенной нотации языка.

Этапы спецификации синтаксиса языка программирования:

- Выделение множества типов синтаксических конструкций.
- Описание множества правил построения синтаксических конструкций, представляемых в виде утверждений следующей формы: *если <часть 1> то <часть 2>*, где *часть 1* содержит знаки типов синтаксических конструкций, *часть 2* содержит выражения, включающие знаки типов синтаксических конструкций, элементарных синтаксических конструкций и операторов конструирования составных синтаксических конструкций.
- Выделение множества типов отношений, заданных на множестве типов синтаксических конструкций.
- Выделение множества типов элементарных синтаксических конструкций, соответствующих задаваемой нотации языка.
- Выделение множества утверждений, необходимых для описания правил построения синтаксических конструкций в задаваемой нотации.

В соответствии со свойствами языков программирования, предназначенных для обработки знаний, были выделены следующие основные типы синтаксических конструкций: программа, оператор, переменная, константа. Операторы являются важным синтаксическим элементом языков программирования, ориентированных на обработку знаний. Операторы задают полное описание действий, которые необходимо выполнить в памяти. Кроме этого, были выделены вспомогательные понятия: тип компонента, тип обрабатываемых данных, тип оператора, атрибут. Введем множество C_{syntax}:

C_{syntax} = {программа, оператор, переменная,

константа, тип оператора, тип компонента, тип обрабатываемых данных, атрибут}.

Семантика ключевых узлов представлена в таблице 1.

Таблица 1 – Семантика ключевых узлов, используемых для описания типов синтаксических конструкций

Ключевой узел	Семантика ключевого узла
программа	знак множества всех синтаксически корректных текстов языка программирования
оператор	знак множества всех операторов, которые могут быть использованы в текстах языка программирования
переменная	знак множества переменных программ
константа	знак множества констант программ
тип оператора	знак множества, элементами которого являются все возможные типы операторов
тип компонента	знак множества, элементами которого являются все возможные типы компонентов, входящих в синтаксические конструкции
тип обрабатываемых данных	знак множества, элементами которого являются все возможные типы данных или структуры данных доступные для обработки синтаксическим элементом
атрибут	знак множества, элементами которого являются все возможные атрибуты синтаксических конструкций и их компонентов

S_{syntax} включает множество логических утверждений, заданных для всех типов синтаксических конструкций. Логические утверждения задают множество правил, которые определяют правильность построения синтаксических конструкций и их иерархию, завершающуюся конструкцией самого верхнего уровня – программой. Знак логического утверждения связывается со знаком типа синтаксической конструкции отношением *Утверждения синтаксиса**. Правильность конструкции проверяется путем подстановки в утверждение значений аргументов.

На множестве ключевых понятий задаются отношения, описывающие общие теоретико-множественные свойства ключевых понятий и связывающие описание синтаксиса языка в формальную модель. Введем множество R_{syntax}, включающее ключевые узлы отношений:

R_{syntax} = {синоним*, пояснение*, разбиение*, пример*, область значений*, обрабатываемые данные*, компонент*, утверждения синтаксиса*}.

Семантика ключевых узлов отношений представлена в таблице 2.

Таблица 2 – Семантика ключевых узлов отношений, используемых для описания синтаксиса

Отношение	Семантика отношения
синоним*	связывает ключевое понятие с множеством синонимов
пояснение*	связывает ключевое понятие с толкованием на других языках
разбиение*	связывает ключевое понятие с множеством его непересекающихся подмножеств
пример*	связывает ключевое понятие с множеством примеров
область значений*	связывает ключевое понятие с множеством возможных значений, в случае наличия ограничений
обрабатываемые данные*	связывает ключевое понятие с множеством обрабатываемых типов синтаксических конструкций
компонент*	связывает ключевое понятие с множеством компонентов, являющихся его структурными частями
утверждения синтаксиса*	связывает знак синтаксической конструкции с множеством правил ее построения и накладываемых ограничений

Ключевые узлы, используемые для описания нотации или множества нотаций языка программирования в базе знаний, зависят от конкретного языка и вводятся при реализации help-системы.

1.2. Описание денотационной семантики

Семантика позволяет описать значения всех синтаксически правильных конструкций языка программирования. Денотационный метод описания семантики основывается на выделении функциональных доменов языка, оперирующих состояниями программы, и придания им значений. Множество состояний определяется преобразованиями абстрактной памяти в процессе исполнения синтаксических конструкций. Таким

образом, денотационная семантика языка программирования рассматривается как множество функций определяющих состояние абстрактной памяти после исполнения синтаксических конструкций относительно исходного состояния абстрактной памяти. Денотационная семантика языка программирования задается функцией

$$F : P \to [S \to S],$$

где P – синтаксическая область программ, S – семантическая область состояний [Ильичева, 2003].

Языки для описания денотационной семантики тесно связаны с λ-исчислением и в качестве базисных используют операции λ-исчисления [Mitchell, 1996], [Shmidt, 1986], [Ильичева, 2003], [Филд и др., 1993], [Кораблин, 1992]. В данной работе методы денотационной семантики предлагается использовать для отображения синтаксиса языка программирования в теоретико-множественные объекты и объекты логики предикатов первого порядка. Описываемый в работе язык $L_{program}$ использует теоретико-множественное описание данных и соответствующие логико-предикатные механизмы для их обработки. Порядок построения формальной модели денотационной семантики языка программирования заимствован из метода, представленного в работе [Ильичева, 2003]. Метод включает следующие шаги:

- выделение множества синтаксических доменов, характеризующих основные типы синтаксических конструкций;
- выделение множества семантических доменов, описывающих множество значений, которыми оперируют синтаксические конструкции;
- выделение множества семантических функций, которые отображают синтаксические конструкции языка программирования в соответствующие семантические домены;
- выделение множества семантических уравнений, которые отображают изменение состояний шаблонов синтаксических конструкций.

Синтаксическим доменам соответствует множество типов основных синтаксических конструкций языка программирования, выделенных в предыдущем разделе:

- константа – множество всех возможных констант;
- переменная – множество всех возможных переменных;
- оператор – множество всех возможных операторов;
- программа – множество всех возможных программ.

Множество семантических доменов включает множество состояний памяти, являющихся результатом выполнения синтаксических конструкций. Во многих языках программирования состояние памяти фиксируется значением переменных, и таким образом, множество семантических доменов является множеством возможных значений переменных. Следовательно, простыми семантическими доменами являются:

- значения переменных (value);
- отсутствие значений (undefined).

На основании простых доменов строятся составные семантические домены:

- абстрактная память (store).

Кроме этого, в качестве семантического домена будем использовать ошибку выполнения (error), в случае неуспешного завершения выполняемого действия.

Введем множество $C_{semantics}$:

$C_{semantics}$ = {значение переменной, абстрактная память, отсутствие значения, ошибка выполнения}.

Для каждого синтаксического домена выделяется одна семантическая функция, устанавливающая его смысл. Множество семантических функций описано в таблице 3.

Таблица 3 – Семантические функции

Синтаксический домен	Семантический домен	Функция
константа	значение переменной	$value$
переменная	значение переменной, отсутствие значения	$\text{var}iable \to [value \times undefined]$
оператор	абстрактная память, значение переменной, отсутствие значения, ошибка	$operator \to [[store \to \text{var}iable, store] \times error]$
программа	абстрактная память	$program \to [store \to store]$

Для каждого синтаксического шаблона задается семантическое уравнение, которое отражает действие функций. Уравнения задаются логическими средствами. Уравнение (1) определяет смысл константы программы. Смыслом константы является сама константа, изменение состояния памяти не происходит.

$$Eq(константа)value = value. \qquad (1)$$

Уравнение (2) показывает, что смыслом переменной является отношение значение*, связывающее знак переменной и элемент, являющийся ее значением. В случае отсутствия значения переменной отсутствует отношение значение*.

$$Eq(переменная)value = \\ < переменная, value > \in значение *. \quad (2)$$

Уравнение (3) задает семантику операторов. Выполнение оператора приводит к изменению значений переменных и изменению состояния памяти, возможно возникновение ситуации, при которой вырабатывается ошибка. Следовательно, смыслом операторов являются: измененные значения переменных и состояние памяти или ошибка выполнения.

$$Eq(оператор)store = для \forall operator \\ если \langle store \rangle \\ то \langle Eq(переменная)value, store_{new} \rangle. \quad (3)$$

Смыслом программы является состояние памяти. Программа изменяет состояние памяти, поэтому значением программы является функция от текущего состояния памяти до нового. Семантикой программы также можно назвать конечные значения переменных.

$S_{semantics}$ включает множество семантических уравнений. Знак семантического уравнения связывается со знаком типа синтаксической конструкции отношением *Утверждения денотационной семантики**. Данное отношение связывает множество синтаксических доменов с семантическими доменами посредствам уравнения. Введем множество $R_{semantics}$:

$R_{semantics}$ = {утверждения денотационной семантики*}

1.3. Описание операционной семантики

Описание операционной семантики для языков программирования, ориентированных на обработку знаний, представляет собой логическую процедуру построения трансляции программ и отдельных синтаксических конструкций. Трансляция строится на некотором метаязыке, содержащем набор инструкций, являющихся простейшими действиями абстрактной машины. В настоящей работе метаязыком является $L_{program}$.

Т.к. разрабатываемый язык $L_{program}$ является подъязыком sc-кода, то наиболее простым способом описания операционной семантики различных языков, ориентированных на обработку знаний, является переход от способов представления знаний и программ к их представлению в виде семантически эквивалентных sc-конструкций. Описываемый язык приводится к некоторому графовому sc-языку. Такому языку ставится в соответствие своя абстрактная sc-машина. Эквивалентность приведенного языка описана в [Голенков и др., 2001b]. В качестве базовой абстрактной sc-машины предлагается scp-машина. Переход от интерпретируемой абстрактной sc-машины произвольного вида к интерпретирующей scp-машине описан в [Голенков и др., 2001b]. Таким образом, $L_{program}$ содержит ключевые узлы для описания scp-машины. При реализации help-системы он дополняется необходимыми ключевыми узлами.

Рассмотрим абстрактную scp-машину. Абстрактная scp-машина задается запоминающей средой (sc-памятью), в которой хранятся перерабатываемые данные, и коллективом агентов (операций) над памятью, переводящих ее из состояния в состояние. Задачами памяти являются хранение программ, а также данных, обрабатываемых программами, и предоставление доступа к ним. Ключевыми понятиями для описания sc-памяти являются хранимые данные и операции, через которые описывается поведение scp-машины. Простейшими данными, хранящимися в sc-памяти, являются sc-элементы. Тип sc-элемента задается структурным типом, константностью и позитивностью. Структура данных для хранения sc-элемента зависит от реализации sc-памяти. В данной работе реализация не описывается, поэтому для описания данных в $L_{program}$ включаем только ключевой узел *sc-элемент*.

Агенты содержат описание методов трансляции правильных синтаксических конструкций описываемого языка в последовательность действий абстрактной scp-машины. Процесс выполнения операции описывается понятием scp-процесс. Для описания текущего состояния scp-процесса в $L_{program}$ включаем ключевые узлы: scp-процесс, тип состояния scp-процесса, ошибка. Простейшими операциями scp-машины, в которые отображается операционная семантика транслируемых синтаксических конструкций, являются:

- Генерация sc-элемента (genEl) – операция генерирует sc-элемент в sc-памяти, тип задается атрибутами. В случае генерации sc-дуги указываются инцидентные элементы. Операции передаются атрибуты, инцидентные элементы (в случае необходимости).
- Удаление sc-элементов (eraseEl) – операция удаления sc-элемента из sc-памяти. Для выполнения операции sc-элемент должен быть определен. Операции передается sc-элемент.
- Поиск sc-элементов или конструкций состоящих из sc-элементов по заданному шаблону (searchEl, searchEls) – операция осуществляет поиск sc-элемента или sc-элементов в sc-памяти по указанным атрибутам. Операции передаются атрибуты.
- Поиск по содержимому sc-элементов (searchContentEl) – операция осуществляет поиск sc-элемента в sc-памяти по указанному содержимому. Операции передаются атрибуты, указывающие тип sc-элемента, и содержимое.
- Преобразование содержимого sc-элементов (convContentEl) – операция осуществляет преобразование содержимого к заданному значению. Для выполнения операции должен быть определен sc-элемент с установленным

содержимым. Операции передается sc-элемент и содержимое.

- Проверка типа sc-элементов (ifTypeEl) – операция проверяет соответствие типа sc-элемента переданным атрибутам. Операции передается sc-элемент и атрибуты.
- Преобразование типа sc-элементов (convTypeEl) – операция преобразовывает sc-элемент к указанному типу. Операции передается sc-элемент и атрибуты.

Дополним множество $C_{semantics}$:

$C_{semantics} = C_{semantics} \cup$ {sc-элемент, scp-процесс, тип состояния scp-процесса, ошибка, генерация sc-элемента, удаление sc-элемента, поиск sc-элемента, поиск sc-элементов по шаблону, поиск по содержимому sc-элемента, преобразование содержимого sc-элемента, проверка типа sc-элемента, преобразование типа sc-элемента}.

Семантика подмножества ключевых узлов $C_{semantics}$ представлена в таблице 4.

Таблица 4 – Семантика ключевых узлов, используемых для описания операционной семантики

Ключевой узел	Семантика ключевого узла
sc-элемент	знак множества всех элементов, хранящихся в sc-памяти
scp-процесс	знак множества всех scp-процессов, каждый из которых выполняется одной из операций scp-машины
тип состояния scp-процесса	знак множества типов состояний scp-процесса
ошибка	знак множества ошибок, возникающих при выполнении sc-процесса
операция	знак множества простейших операций, вызываемых при выполнении синтаксических конструкций
генерация sc-элемента	знак множества операций генерации sc-элемента
удаление sc-элемента	знак множества операций удаления sc-элемента
поиск sc-элемента	знак множества операций поиска sc-элемента
поиск sc-элементов по шаблону	знак множества операций поиска sc-элементов по шаблону
поиск по содержимому sc-элемента	знак множества операций поиска по содержимому sc-элемента
преобразование содержимого sc-элемента	знак множества операций преобразования содержимого sc-элемента
проверка типа sc-элемента	знак множества операций проверки типа sc-элемента
преобразование типа sc-элемента	знак множества операций преобразования типа sc-элемента

Каждая операция характеризуется именем, условием инициирования, входными/выходными данными, трансляцией. Дополним множество $R_{semantics}$:

$R_{semantics} = R_{semantics} \cup$ {условие инициирования*, входные данные*, выходные данные*, трансляция*}.

Семантика подмножества ключевых узлов отношений $R_{semantics}$ представлена в таблице 5.

Таблица 5 – Семантика ключевых узлов отношений, используемых для описания операционной семантики

Отношение	Семантика отношения
условия инициирования*	связывает операцию с шаблоном ее инициирования
входные данные*	связывает операцию с множеством sc-элементов, являющимися входными данными операции
выходные данные *	связывает операцию с множеством sc-элементов, являющимися выходными данными операции
трансляция*	связывает операцию с последовательностью простейших операций, отображающих алгоритм ее выполнения

Таким образом, для описания операционной семантики требуется выделение ключевых узлов, обозначающих типы синтаксических конструкций. Необходимые ключевые узлы были включены ранее. Требуются ключевые узлы, описывающие переход от используемых способов представления знаний и программ к их представлению в виде семантически эквивалентных sc-конструкций. Требуемые ключевые узлы определяются при реализации help-системы и зависят от описываемого языка программирования. Требуются ключевые узлы для описания scp-машины.

2. Семантический язык для описания интегрированной среды разработки программ

Интегрированная среда разработки программного обеспечения состоит из

инструментов, обеспечивающих разработку программ и среду существования программ, а также вспомогательных средств, используемых для управления разработкой программ. Обычно она ориентирована на определенный язык программирования или языки программирования одной методологии. Базовыми функциями сред являются: управление кодом программ или проектов, хранение версий проектов и его различных компонентов, сборка проектов, подключение и настройка транслятора, отладка кода, получение справочной информации. Обзор сред, используемых для проектирования программ, ориентированных на обработку знаний, показал, что они обладают свойствами и функциональностью сред для проектирования программ на традиционных языках.

Для формального описания интегрированной среды предлагается построить ее компонентную модель. Каждый компонент определяется своей функциональностью и может включаться в состав среды или отсутствовать. Компонент представляется своей формальной моделью. Таким образом, формальной моделью интегрированной среды разработки программного обеспечения является ее компонентная модель и совокупность формальных моделей каждого компонента. Формальная модель среды описывается на семантическом языке L_{ide}. L_{ide} задается объединением множеств

$$L_{ide} = C_{ide} \cup S_{ide} \cup R_{ide},$$

где C_{ide} – множество ключевых узлов, предназначенных для описания формальных моделей компонентов интегрированных сред; S_{ide} – множество утверждений, предназначенных для описания компонентов интегрированных сред; R_{ide} – множество отношений, определенных на множествах C_{ide} и R_{ide}.

На основании базовых функций выделим основные компоненты, включаемые в интегрированные среды, и введем множество C_{ide}:

C_{ide} = {редактор исходного кода, транслятор, компоновщик, отладчик, интерфейс, справочная система}.

Рассмотрим формальные модели каждого компонента и дополним язык C_{ide}.

Редактор предназначен для управления текстами программ или проектов. Редактор может поддерживать один или несколько языков программирования, обнаруживать синтаксические ошибки пользователя по мере ввода текста, показывать всплывающие подсказки, осуществлять дополнение кода. Среда программирования включает один или несколько редакторов для объектов различного вида (например, различные нотации языка могут редактироваться в различных редакторах). При использовании линейной записи текстов языка редактор обеспечивает следующие функции: подсветка синтаксиса, автоматическое указание ошибок, форматирование текста, загрузка шаблонов синтаксических конструкций языка, автоматическое дополнение кода, всплывающие подсказки при наведении на синтаксический элемент (помощник при работе с содержимым исходного файла). При использовании графической записи текстов языка редактор обеспечивает следующие функции: подсветка синтаксиса, загрузка шаблонов синтаксических конструкций языка, всплывающие подсказки при наведении на синтаксический элемент. Дополним язык C_{ide}:

$C_{ide} = C_{ide} \cup$ {язык программирования, нотация языка программирования, подсветка синтаксиса, автоматическое указание ошибок, форматирование текста, загрузка шаблона, дополнение кода, всплывающие подсказки}.

Транслятор выполняет преобразование программ с одного языка в эквивалентные им программы на другом языке. В случае языка программирования – на язык понятный компьютеру. Реализация транслятора не является задачей данной работы, поэтому ограничимся описанием операционной семантики языка программирования, представленной ранее. Среда включает только функции, используемые для запуска или остановки работы транслятора. Дополним язык C_{ide}:

$C_{ide} = C_{ide} \cup$ {запуск процесса, завершение процесса}.

Компоновщик формирует проект, собирая необходимые файлы-компоненты (программы, библиотеки, файлы данных и др.) и редактируя перекрестные ссылки. Среда вызывает компоновщик при сборке проекта, при запуске программ на выполнение, при обновлении ресурсов и сохранении измененных ресурсов на диск. Дополним язык C_{ide}:

$C_{ide} = C_{ide} \cup$ {библиотека, файл данных, ресурс}.

Отладчик используется для обнаружения ошибок в программе или причин их появления при пошаговом просмотре результатов ее выполнения. Для описания отладчика необходимо представить модель процесса трансляции программы и дополнить ее параметрами отладки. Основными функциями отладчика являются:

- управление выполняемым процессом: запуск, завершение, выполнение в пошаговом режиме, приостановка, возобновление, создание контрольных точек останова;
- модификация памяти: изменение значений переменных, системных структур данных;
- представление пользователю информации о состоянии памяти и сведений о ходе выполнения процесса.

Реализация отладчика зависит от каждого конкретного языка и от средств, используемых для его реализации. Традиционно выполняемому процессу ставится в соответствие микропрограмма,

которая получает сведения о его выполнении и позволяет его выполнять в пошаговом режиме. Далее микропрограмму в работе будем называть трассировщиком. Трассировщик запускает процесс или присоединяется к существующему процессу и имеет доступ ко всем данным процесса и параметрам отладки, что позволяет ему выполнять все вышеперечисленные функции. Пошаговый режим отладки выполняется в соответствии с управляемой пользователем спецификацией шагов: выполнить одну инструкцию с заходом в функцию, выполнить одну инструкцию без захода в функцию, выполнить выход из функции, продолжить выполнение программы до следующей точки останова, продолжить выполнение программы до курсора, продолжить выполнение программы до выполнения заданного условия. Контрольные точки останова загружаются в память вместе с текстом программы и связываются с синтаксической конструкцией. Контрольные точки представляются в виде структуры данных, содержащей всю необходимую информацию. Например, структура данных может содержать: тип точки останова, связь со строкой (в случае линейной записи программ) или связь с ключевым узлом (в случае графического представления программ), маркер ресурсов для сохранения информации о контрольных точках между сессиями и др.

Отображение текущего состояния памяти обеспечивается компонентами пользовательского интерфейса, настраиваемыми пользователем. Возможно отображение содержимого областей памяти, значений переменных, стека вызова процедур. Дополним язык C_{ide}:

$C_{ide} = C_{ide} \cup$ {процесс, проект, программа, память, трассировщик, режим отладки, точка останова, тип точки останова}.

Интерфейс включает настройки для различных режимов работы. Например, наиболее часто встречаются следующие режимы работы: режим разработки программ, режим отладки, режим выполнения. Настройки включают конфигурирование вида и расположения используемых инструментов и пунктов меню, настройку внешнего вида иконок и количества иконок при различных режимах работы, настройки используемого транслятора и настройки редактора кода. Интерфейс обеспечивает работу мастеров при создании проектов или отдельных файлов. Дополним язык C_{ide} ключевыми узлами, обеспечивающими описание интерфейса:

$C_{ide} = C_{ide} \cup$ {режим разработки, режим отладки, режим выполнения, вид инструментального средства, проект, файл, мастер создания проекта}.

Справочная система представляет собой консультационную программную систему по технологии разработки программного обеспечения. Существующие инструментальные средства включают различные виды справочных систем. Поэтому в данной работе не предлагается единая модель для описания используемых справочных систем, а предлагается модель их построения. В качестве справочной системы может использоваться предлагаемая в работе help-система, либо ее справочная подсистема.

S_{ide} включает множество логических утверждений, описывающих особенности работы инструментального средства.

R_{ide} включает множество отношений, описывающих общие теоретико-множественные свойства ключевых узлов и связывающих ключевые понятия множеств C_{ide} и S_{ide} в единую формальную систему. Введем множество R_{ide}:

R_{ide} = {синоним*, пояснение*, разбиение*, компонент*, утверждения*, поддерживаемые нотации языка*, шаблон*, используемый транслятор*, выполняемые действия*}.

Семантика подмножества ключевых узлов отношений R_{ide} представлена в таблице 6.

Таблица 6 – Семантика ключевых узлов отношений, используемых для описания интегрированной среды разработки программ

Отношение	Семантика отношения
синоним*	связывает ключевое понятие с множеством синонимов
пояснение*	связывает ключевое понятие с толкованием на других языках
разбиение*	связывает ключевое понятие с множеством его подмножеств
компонент*	связывает ключевое понятие с множеством его компонентов, файлов-компонентов
утверждения*	связывает ключевое понятие с множеством утверждений
поддерживаемые нотации языка*	связывает ключевое понятие редактора исходного кода с множеством поддерживаемых нотаций языка (языков) программирования
шаблон*	связывает ключевое понятие с множеством шаблонов, доступных для дополнения
используемый транслятор*	связывает ключевое понятие (знак нотации языка программирования) со знаком используемого транслятора или множеством знаков используемых трансляторов
выполняемые действия*	связывает ключевое понятие с действием, которое выполняется при его активации в инструментальной среде разработки программ

3. Семантический язык для описания методики проектирования программ, ориентированных на обработку знаний

Проектирование программ требует специальных методов. Существует два основных класса методов, используемых как для разработки программ на традиционных языках, так и для разработки на языках, ориентированных на обработку знаний. Классическим методом разработки программ является метод водопада, который включает определение требований, проектирование, кодирование, интеграцию, тестирование, внедрение и поддержку. Наиболее популярным является итеративный метод. Итеративный метод представляет собой многократный проход этапов разработки с запланированным улучшением результата. Каждый этап, называемый итерацией, является мини-проектом фиксированной длительности разрабатываемой системы. Существует множество примеров использования данного метода: быстрая разработка приложений (Rapid application development, RAD), экстремальное программирование (Extreme programming, XP), Microsoft Solution Framework (MSF), Rational Unified Proces (RUP), Dynamic Systems Development Method (DSDM), SADT (IDEFx), Scrum и др.

На основании анализа методов выделены их общие свойства. В работе представлена обобщенная формальная модель некоторых методов проектирования программ и формальная модель эволюционной методики проектирования интеллектуальных систем, которая описана в [Электронный ресурс], [Гулякина и др., 2011], [Голенков и др., 2012]. Методы тесно связаны со средствами проектирования программ, поэтому при их описании следует учитывать перспективы развития средств и их влияние на развитие методов. Для описания методики проектирования программ в работе предлагается семантический язык

$$L_{method} = C_{method} \cup R_{method},$$

где C_{method} – множество ключевых узлов, предназначенных для описания методики проектирования программ; R_{method} – множество отношений, определенных на множестве ключевых узлов.

На основании анализа используемых методов выделим общие ключевые понятия для их описания и введем множество C_{method}:

C_{method} = {методология, метод, этапы проектирования, требования, проектирование, кодирование, интеграция, тестирование, внедрение, поддержка, прототип, инструментальное средство, язык программирования, язык представления знаний}.

Формальная модель эволюционной методики проектирования интеллектуальных систем включает методику проектирования баз знаний, методику проектирования машины обработки знаний, методику проектирования пользовательских интерфейсов, методику проектирования интеллектуальных решателей задач. Дополним множество C_{method}:

$C_{metod} = C_{method} \cup$ {методика проектирования базы знаний, методика проектирования операций, методика проектирования пользовательского интерфейса, методика проектирования интеллектуального решателя задач}.

Каждая методика применяется на одном из этапов построения интеллектуальной системы. На первом этапе проектируется база знаний интеллектуальной системы. Методика построения базы знаний описана в [Электронный ресурс]. Она включает следующие шаги:

1. Разработка тестового сборника вопросов (спецификации вопросов).
2. Запись ответов на вопросы в текстовом и формальном виде.
3. Тестирование стартовой версии базы знаний.
4. Выделение набора объектов, входящих в состав базы знаний: классов объектов, отношений, связок, набора основных логических высказываний (утверждений) для описания свойств объектов.
5. Представление выделенных понятий на формальном языке.
6. Верификация базы знаний.
7. Анализ и уточнение распределения понятий по логическим уровням.
8. Логико-дидактическая структуризация базы знаний: декомпозиция и упорядочение выделенных разделов.
9. Анализ полноты базы знаний.
10. Тестирование полученной версии базы знаний.

В результате получаем первый прототип интеллектуальной системы, который включает разработанную базу знаний, базовую информационно-поисковую машину и базовый пользовательский интерфейс. Для описания формальной модели методики проектирования баз знаний дополним множество C_{method} множеством шагов построения базы знаний:

$C_{method} = C_{method} \cup$ {спецификация вопросов, ответы на вопросы, тестирование стартовой версии, выделение классов объектов, выделение отношений, выделение связок отношений, выделение утверждений, представление понятий на формальном языке, верификация базы знаний, распределение понятий по логическим уровням, структуризация базы знаний, анализ полноты базы знаний, тестирование базы знаний, базовая информационно-поисковая машина, базовый пользовательский интерфейс}.

На втором этапе интеллектуальная система дополняется множеством специализированных операций машины обработки знаний. Методика проектирования операций:

1. Выделение множества операций (на основании анализа спецификации вопросов).
2. Спецификация каждой операции на формальном языке.
3. Реализация операций на языке программирования.
4. Отладка операций.
5. Тестирование полученной версии машины обработки знаний.

В результате получаем второй прототип интеллектуальной системы, который включает разработанную базу знаний, специализированную машину обработки знаний и базовый пользовательский интерфейс. Для описания формальной модели машины обработки знаний дополним множество C_{method}:

$C_{method} = C_{method} \cup$ {операция, спецификация операции, отладка операции, тестирование машины обработки знаний}.

На третьем этапе проектируется предметно-ориентированный пользовательский интерфейс. Методика проектирования пользовательского интерфейса описана в работе [Корончик, 2012]. Процесс проектирования включает:

1. Спецификацию интерфейса: список решаемых задач, описание внешних языков представления знаний.
2. Создание задачно-ориентированной декомпозиции пользовательского интерфейса.
3. Разработка компонентов, каждый из которых является подсистемой.
4. Тестирование пользовательского интерфейса.

Третий прототип интеллектуальной системы является реализованной системой, удовлетворяющей требованиям разработки. Дополним множество C_{method}:

$C_{method} = C_{method} \cup$ {интерфейс, спецификация интерфейса, декомпозиция интерфейса, тестирование интерфейса}.

На четвертом этапе интеллектуальная система дополняется решателем задач, который осуществляет генерацию ответов на заданные пользователем вопросы в случае, если ответы отсутствуют в текущем состоянии базы знаний. Методика проектирования описана в [Заливако и др., 2012]. Компонентами решателя задач являются операции, поэтому методика проектирования сводится к методике проектирования операций.

4. Семантический язык для описания методики обучения проектированию программ

Интеллектуальная help-система может использоваться в качестве компьютерного средства обучения (далее – КСО). В [Башмаков и др., 2003] выделены основные педагогические задачи, которые должны решать КСО:

- начальное ознакомление с предметной областью, освоение базовых понятий и концепций;
- базовая подготовка на разных уровнях глубины и детальности;
- выработка умений и навыков решения типовых практических задач;
- выработка умений анализа и принятия решений в нестандартных проблемных ситуациях;
- развитие способностей к определенным видам деятельности;
- восстановление знаний, умений и навыков (для редко встречающихся ситуаций, задач и технологических операций);
- контроль и оценивание уровней знаний и умений.

Для решения вышеперечисленных задач база знаний интеллектуальной help-системы должна содержать формальные модели методики обучения проектированию программ и стратегии обучения программированию. Следовательно, семантический язык описания методики обучения программированию $L_{training}$ является объединением множеств

$$L_{training} = C_{training} \cup R_{training},$$

где $C_{training}$ – множество ключевых узлов, предназначенных для описания формальных моделей методики обучения проектированию программ и стратегии обучения; $R_{training}$ – множество отношений, определенных на множестве ключевых узлов.

Рассмотрим методику обучения как совокупность следующих взаимосвязанных иерархических компонентов: целевой, содержательный, деятельностный, результативный, управленческий. Введем множество $C_{training}$:

$C_{training}$ = {целевой компонент, содержательный компонент, деятельностный компонент, результативный компонент, управленческий компонент}.

Целевой компонент содержит формулировку целей и задач обучения. Дополним множество $C_{training}$:

$C_{training} = C_{training} \cup$ {цель обучения, задача обучения}.

Содержательный компонент включает содержание технологии и теоретическую часть, позволяющую предоставить пользователю целостную картину процесса проектирования программ. Теоретическая часть представляется в виде онтологии технологии. Так как в данной работе рассматривается технология проектирования интеллектуальных систем, то содержательный компонент дополняется теорией для осуществления предварительного этапа обучения. Предварительный этап формирует у разработчиков понимание отличия традиционного программирования от программирования

интеллектуальных систем. Приведем описание на примере разработки теоретико-графовых задач. Для этого в базу знаний включим формализацию теоретико-графовых алгоритмов и запись их в виде программ на различных языках. Например:

- на процедурном языке программирования высокого уровня, ориентированном на обработку семантических сетей;
- на базовом языке программирования, ориентированном на обработку семантических сетей;
- на традиционном языке программирования с использованием специализированной библиотеки моделирования графодинамической памяти.

Далее теоретическая часть содержит описание эволюционных этапов проектирования интеллектуальных систем. Дополним множество $C_{training}$:

$C_{training} = C_{training} \cup$ {технология программирования, язык программирования, алгоритм, этап проектирования интеллектуальной системы}.

Деятельностный компонент направлен на формирование у пользователя практических навыков в области разработки интеллектуальных систем и приобретения опыта применения теоретических знаний при проектировании программ. Компонент может осуществлять решение задач и объяснять способы их решения. Дополним множество $C_{training}$:

$C_{training} = C_{training} \cup$ {задача, способы решения задачи}.

Результативный компонент обеспечивает генерацию тестовых и практических заданий, проведение тестирования и интерпретацию результатов. Компонент может проводить оценку знаний и умений пользователя. Дополним множество $C_{training}$:

$C_{training} = C_{training} \cup$ {тестовое задание, тестирование, формы проведения тестирования}.

Управленческий компонент несет организационную, обучающую и контролирующую функции, организует адаптивный диалог с пользователем. Основной задачей компонента является обеспечение пользователю помощи, соответствующей ситуации, и выдача рекомендаций с учетом истории его взаимодействия с системой, т.е. осуществление поддержки принятия решений при разработке программного обеспечения для конкретного разработчика программ. Управленческий компонент определяет стратегию обучения для минимизации времени обучения. Дополним множество $C_{training}$:

$C_{training} = C_{training} \cup$ {стратегия обучения}.

Наряду с иерархией понятий технологии программирования используются онтологии знаний, умений, навыков пользователей, которые представлены в формальной модели разработчика программ. Формальная модель задается в виде шаблона, который наполняется в процессе работы пользователя с системой и фиксируется для каждого конкретного пользователя. Дополним множество $C_{training}$:

$C_{training} = C_{training} \cup$ {модель пользователя}.

Кроме этого, в связи с высокими темпами обновления технологий разработки программ, появления новых языков программирования, развития новых инструментальных средств разработки программ меняются и подходы к методике обучения проектированию программ. Это необходимо учитывать при проектировании help-системы.

Заключение

В работе представлены семантические языки представления знаний, являющиеся подъязыками sc-кода. Семантические языки предоставляют средства для описания технологии разработки программ, ориентированных на обработку знаний, в базе знаний интеллектуальной help-системы. Предлагаемые языки позволяет явно представить семантическую структуру предметной области, что облегчает реализацию семантической навигации по всей структуре информации, а также ассоциативного поиска информации. Также язык позволяет строить спецификацию технологии, используемую для формирования ответов на вопросы пользователя, для обучения пользователя, контроля знаний и умений пользователя. Построенная спецификация представляет собой модель транслятора языка программирования, инструментального средства, и может использоваться для проектирования и реализации данных компонентов технологии.

Преимуществом семантических языков является предоставление возможности интеграции различных компонентов, построенных с их помощью и с помощью других sc-языков. Благодаря чему можно объединять компоненты, содержащие знания по различным технологиям проектирования программного обеспечения, в одну help-систему.

Предлагаемые языки являются легко расширяемыми, поскольку их всегда можно пополнить множеством требуемых ключевых узлов.

Библиографический список

[Голенков и др., 2012] Голенков, В.В. Графодинамические модели параллельной обработки знаний: принципы построения, реализации и проектирования / В.В.Голенков, Н.А.Гулякина // Материалы международной научно-технической конференции «Открытые семантические технологии проектирования интеллектуальных систем» – Минск:БГУИР, 2012. – с. 23-52

[Электронный ресурс] Открытая семантическая технология компонентного проектирования интеллектуальных систем [Электронный ресурс]. – Электронные данные. – Режим доступа: http://ostis.net/mediawiki/index.php/.

[Голенков и др., 2001a] Голенков, В.В. Представление и обработка знаний в графодинамических ассоциативных

машинах. Монография / В.В. Голенков, О.Е. Елисеева, В.П. Ивашенко и др. Под ред. В.В. Голенкова. – Мн.: БГУИР, 2001. – 412 с.

[Себеста, 2001] Себеста, Роберт У. Основные концепции языков программирования, 5-е изд.: Пер. с англ. / Роберт У. Себеста. – М.: Издательский дом «Вильямс», 2001. – 672 с.

[Пратт и др., 2002] Пратт, Т. Языки программирования: разработка и реализация. 4-е изд.: Пер. с англ. / Т.Пратт., М.Зелковиц. Под общей ред. А.Матросова. – СПб.: Питер, 2002. – 688 с.

[Вольфенгаген, 2001] Вольфенгаген, В.Э. Конструкции языков программирования. Приемы описания / В.Э. Вольфенгаген. – М.: АО «Центр ЮрИнфоР», 2001. – 276 с.

[Mitchell, 1996] Mitchell, John C. Foundations for programming languages / John C. Mitchell. – The MIT Press, 1996. – 845 с.

[Shmidt, 1986] Shmidt, David A. Denotational semantics: a methodology for language development / David A. Shmidt. – Allyn and Bacon, 1986. – 331 с.

[Ильичева, 2003] Ильичева, О.А. Формальное описание семантики языков программирования. Электронный учебник / О.А. Ильичева. – Ростов на Дону: ЮФУ, 2007. – 223 с.

[Филд и др., 1993] Филд, А. Функциональное программирование: Пер. с англ. / А.Филд, П. Харрисон. – М.: Мир, 1993. – 637 с.

[Кораблин, 1992] Кораблин, Ю.П. Семантика языков программирования: учебное пособие / Ю.П. Кораблин. Под ред. – М.: МЭИ, 1992. – 102 с.

[Голенков и др., 2001b] Голенков, В.В. Программирование в ассоциативных машинах / В.В.Голенков, Г.С.Осипов, Н.А.Гулякина и др. – Мн.: БГУИР, 2001. – 276 с.

[Гулякина и др., 2011] Гулякина, Н.А. Комплексная методика проектирования и обучения проектированию интеллектуальных справочных систем / Н.А.Гулякина, О.В.Пивоварчик // Материалы международной научно-технической конференции «Открытые семантические технологии проектирования интеллектуальных систем». – Минск: БГУИР, 2011. – с. 519 – 522

[Голенков и др., 2012] Голенков, В.В. Графодинамические модели параллельной обработки знаний: принципы построения, реализации и проектирования / В.В.Голенков, Н.А.Гулякина // Материалы международной научно-технической конференции «Открытые семантические технологии проектирования интеллектуальных систем». – Минск: БГУИР, 2012. – с. 23 – 52

[Корончик, 2012] Корончик, Д. Н. Семантические модели мультимодальных пользовательских интерфейсов и семантическая технология их проектирования / Д.Н.Корончик // Материалы международной научно-технической конференции «Открытые семантические технологии проектирования интеллектуальных систем». – Минск: БГУИР, 2012. – с. 339 – 346

[Заливако и др., 2012] Заливако, С. С. Семантическая технология компонентного проектирования интеллектуальных решателей задач / С.С.Заливако, Д.В.Шункевич // Материалы международной научно-технической конференции «Открытые семантические технологии проектирования интеллектуальных систем». – Минск: БГУИР, 2012. – с. 297 – 314

[Башмаков и др., 2003] Башмаков, А. И. Разработка компьютерных учебников и обучающих систем / А. И. Башмаков, И. А. Башмаков — М.: Информационно-издательский дом «Филинъ», 2003.

SEMANTIC LANGUAGES USED FOR DESCRIPTION OF LANGUAGES PROGRAMMING, ORIENTED TO KNOWLEDGE TREATMENT

Pivovarchyk O.V.

**Belarusian State University of Informatics and Radioelectronics, Minsk, Republic of Belarus*

pivovarchyk@tut.by

The knowledge representation semantic languages is described in the article. The semantic languages are used to develop knowledge base of help-system for software developers. The semantic languages are sc-languages and have their extention in the set of key nodes.

INTRODUCTION

It is necessary to create semantic languages for describing a technology software engineering to build the intellectual help-system knowledge base. The semantic language for description of programming language ($L_{program}$), the semantic language for description of integrated development environment (L_{ide}), the semantic language for description of methodology software engineering (L_{method}), the semantic language for description of methodology of training of software engineering ($L_{training}$) are proposed in this article.

MAIN PART

The programming language formal model are presented its specification, which contains language syntax and semantics. $L_{program}$ includes following sets: the set of key nodes and the set of statement for description of programming language syntax, denotational semantics and operational semantics; the set of relation for union the sets of key nodes in the integrated formal system.

In the knowledge base of the integrated development environment componet models are presented. L_{ide} includes following sets: the set of key nodes and the set of statement for description of formal models of integrated development environment components; the set of relation for union the sets of key nodes in the integrated formal system.

L_{method} allows to describe the generalize model of software engineering methods and the model of intellectual systems engineering evolutional methodology. L_{method} includes following sets: the set of key nodes for description of software engineering methodology; the set of relation for union the set of key nodes in the integrated formal system.

In the knowledge base the methodology of training of software engineering are presented formal models of training methods and formal models of training strategies. Consequently, $L_{training}$ includes following sets: the set of key nodes for description the formal models of methods of software engineering training and the training strategies; the set of relation for union the set of key nodes in the integrated formal system.

CONCLUSION

The knowledge representation semantic languages allow to design the technology software engineering semantic model in the knowledge base. The semantic model is based on meaning links. This facilitates implementation of semantic navigation, of associative search, of the formation answer to the user query.

The knowledge representation semantic languages are scalable.