Вісник Харківського національного університету
Серія «Математичне моделювання. Інформаційні технології. Автоматизовані системи управління»
УДК 519.6+681.3.06 № 605 , 2003, с. 90-105

Прикладной язык семантической структуризации научных текстов как некоторая проекция языка Ада

В. О. Мищенко
Харьковский национальный университет им. В.Н. Каразина, Украина

This work proves that it is possible to develop an applied language for semantic structuring of scientific documents (LSS) based on the appropriate projection of the Ada language. (The author introduced the projection concept for languages defined by their formal grammar onto the given nonterminal subset along the given subset of this grammar's rules earlier.) Not equality but absolute correspondence between the syntax of the developed LSS and the syntax of the given projection of the Ada language has been found Thereby, it is possible to realize the developed LSS on the basis of standard information technologies related to Ada.

1. Семантическая структуризация научных текстов и постановка задачи о расширении языков семантической структуризации (ЯСС)

Налаживание работы с информационными системами научных направлений через Интернет весьма актуально, как для представляющих эти направления учёных, так и для целей обучения, особенно по программам магистратуры и аспирантуры. При этом информационные системы направлений, зависящих от быстрого развития исследовательского программного обеспечения, должны обладать определёнными особенностями, рассмотренными в [1]. Настоящая работа относится к вопросам новых специальных информационных технологий, необходимых для создания и эксплуатации информационных систем указанного вида.

Для описания смысловой структуры совокупности научных текстов в рамках информационной системы научного направления было предложено использовать систему связей, построенную по образцу межмодульных связей в программах на языке Ада [1]. В качестве средства моделирования переноса средств структуризации языка программирования на специальные научные тексты было введено понятие проекции языка, определяемого формальной грамматикой, на заданное подмножество нетерминалов вдоль заданного подмножества правил этой грамматики. Реальным примером использования такой модели переноса служит очень простая проекция контекстно-свободного синтаксиса языка Ада, предложенная в [2]. Эта проекция определила модель семантической структуризации информационной системы научного направления для этапа начального наполнения базы данных такой системы. Отметим, что, трактовка проекции существующего языка как средства создания новых специализированных языков не противоречит абстрактному понятию проекции языков [3].

Модель семантической структуризации [2] позволяет отразить связи таких материалов, как статьи и книги, между собой, а также выделить структурные единицы - модули внутри материалов. Актуально и практически важно

расширить модель в первую очередь за счёт добавления средств отражения связей между любыми модулями, включая внутренние модули одного и того же материала. Проекция формального языка (в нашем случае контекстно-свободного синтаксиса языка программирования Ада) легко трактуется как формальный синтаксис для языка разметки текстов (в данном случае научных). При этом определение проекции в [2] ориентировано на то, чтобы интерпретировать контекстные ограничения синтаксиса языка Ада в качестве контекстных ограничений для синтаксиса языка разметки, порождаемого проекцией Ады. Таким образом, полностью определяется синтаксис языка семантической структуризации текстов – ЯСС. Версию такого языка, определённую в [2], обозначим индексом 1.0. Отметим, хотя это не тема данной статьи, что семантика подобных ЯСС в значительной степени должна опираться на статическую семантику исходного языка (языка программирования Ада). А именно, семантика языка, полученного проекцией, должна определяться в тесной связи с интерпретацией его нетерминалов, как понятий, применимых к научным текстам и, насколько возможно, аналогичных понятиям, обозначаемым этими нетерминалами в статической семантике языка Ада.

Если теперь вернуться к ЯСС, определённому в [2] через проекцию, то ясно, что при его расширении в соответствии с нуждами прикладного использования, полученный расширенный ЯСС может и не быть никакой проекцией языка Ада. Это поставило бы под сомнение корректность системы контекстных ограничений расширения (поскольку эта система не вытекала бы из такой системы для заведомо корректного языка) и противоречило бы концепции рассматриваемых информационных систем [1]. Последнее связано с тем, что в базах данных интересующих нас информационных систем научные тексты должны храниться на равных правах с исходными текстами программного обеспечения, и запросы о семантических связях должны выявлять межмодульные и прочие связи независимо от того, идёт ли речь о связях текстовых или программных модулей.

Поэтому ставится задача построения расширенного ЯСС с вышеуказанными пользовательскими качествами так, чтобы правила его формального синтаксиса соответствовали правилам формального синтаксиса языка-прототипа, в данном случае - Ады. Соответствие здесь понимать, как такое, которое даёт возможность однозначного переноса необходимых контекстных ограничений из прототипа в ЯСС.

2. Обоснование корректности ЯСС методом последовательных сравнений с проекциями языка Ада

Воспроизведём и немного расширим понятие проекции языка в смысле [2]. Пусть

$$G = (T, N, S, n_0) \qquad (2.1)$$

– формальная КС-грамматика, обеспечивающая однозначность синтаксического разбора (в терминологии [4]) и определяющая основу синтаксиса для языка P (языка-прототипа ЯСС). Здесь T, N, S – множества, соответственно, терминалов, нетерминалов и грамматических правил, а n_0 – нетерминал. Множество фраз, выводимых из n_0 , обозначим P_0 . Синтаксически правильные фразы языка-

прототипа P отбираются из P_0 с учётом контекстных ограничений, которые в практике широко используемых языков программирования излагаются на естественном языке.

Пусть N_1 - подмножество N , содержащее n_0 . «Ортогональную» проекцию P на N_1 определит подмножество правил S_{N1} , полученное из S исключением тех правил, в которых слева от символа непосредственного вывода «→» фигурируют нетерминалы, не принадлежащие N_1. Исходя из прикладных соображений, можно использовать какую-то другую проекцию на N_1, сужая S_{N1} так, чтобы в оставшемся множестве S_0 для каждого n из N_1 осталось хотя бы одно правило вида n–>α с непустой цепочкой α

Сформируем T_1 , добавляя в T новый терминал @ («замещение»). Множество S_1 образуем, исходя из S_0 , так. Для всякого вхождения в правила, принадлежащие S_0 , какого-то нетерминала n , не принадлежащего N_1 добавляем к правилам из S_0 правило

$$n->@ \tag{2.2}$$

Формально под «проекцией» P_1 языка P «на» N_1 «вдоль» S_1 будем понимать язык, порожденный грамматикой

$$G_{S1} = (T_1, N_1, S_1, n_0) \tag{2.3}$$

Из подобной проекции можно интерпретацией получить язык разметки текстов. Последние представляют собой последовательности фраз «естественного» языка L . Так у нас L - язык научных текстов, допускающий, в частности, и формулы. Интерпретация грамматики (2.2) чисто формальна: символ @ переводится из разряда терминалов в нетерминалы, но добавляются грамматические правила вида

$$@-> l \tag{2.4}$$

для всевозможных фраз l из языка L . Язык, порождаемый такой формальной грамматикой, обозначим $(L_{S1})_0$. Менее формальным, но в практическом смысле также однозначным является построение свода контекстных ограничений, сужающих $(L_{S1})_0$ до практического языка L_{S1} типа интересующего нас ЯСС:

L_{S1} ($(L_{S1})_0$ с контекстными правилами)

Сейчас мы изложим подробности.

Полагая язык-прототип языком программирования, будем считать, что все его контекстные зависимости формулируются в терминах равенства-неравенства терминальных цепочек, подпадающих под такие понятия, как имена и идентификаторы. Также, практически не ограничивая общности, считаем, что все обозначающие эти понятия нетерминалы входят в N_1 . Но тогда соответствующие этим понятиям терминальные цепочки во фразах языка $(L_{S1})_0$ суть цепочки (они из элементов T и L), для которых «равенство» имеет известный смысл. Рассмотрим контекстное правило K языка-прототипа P ,

применимое к цепочке α, выводимой из n_0 посредством правил из S_0 . Пусть K ссылается на равенство или неравенство между собой неких частей μ, ν, … этой цепочки. Эти части подпадают под понятия языка P , и для любой фразы φ языка $(L_{S1})_0$, выведенной из α, части этой фразы, соответствующие μ, ν, … , имеют структуру вида $s_1 s_2 .. s_k$, где s_i являются либо терминалами из T либо фразами из L. Тогда, по сделанному выше замечанию о сравнимости фраз, K осмысленно для φ и позволяет решить, можно ли отнести эту фразу к содержательному языку L_{S1} . В силу предполагавшейся однозначности синтаксического разбора в P_0 , можно по определению положить, что все контекстные зависимости для L_{S1} формируются таким и только таким образом. Содержательный синтаксис ЯСС полностью определён.

Настоящая работа посвящена разработке пригодного к практическому использованию языка семантической структуризации научных текстов следующим методом, который можно будет использовать и в других подобных ситуациях. Отправляясь от некоторой (в данном случае - описанной в [2]) «минимальной» проекции языка-прототипа, добавляем новые (контекстно-независимые по форме) правила для обслуживания расширенным языком новых функций. При этом для каждого нового понятия (нетерминала) ищем из неформальных соображений наиболее близкое к нему понятие (нетерминал) в языке-прототипе. Добавленные правила сравниваем (ориентируясь на их левые части) с теми, которые имеют место для предполагаемых понятий-прообразов, и выясняем, можно ли интерпретировать добавленные правила, как получаемые подходящим проектированием в смысле [2]. Если нет, то проверяем, нельзя ли преодолеть имеющиеся расхождения добавленных правил с «ближайшей» проекцией так, чтобы сохранилась возможность прозрачного переноса правил контекстной зависимости для понятий-прообразов на нововведенные понятия разрабатываемого языка. При новой неудаче пересматриваем соответствие добавляемых новых понятий и понятий прообразов, а в крайнем случае - корректируем добавляемые правила. В целом процедура подходит под известную в математике схему, когда доказательство сложной теоремы строится одновременно с окончательным вариантом формулировки её условия.

Оказалось, что по сравнению с осуществляемыми при проектировании операциями, в нашей практике обнаружилось ещё только два новых приёма образования правил ЯСС, которые не препятствуют переносу контекстных правил из языка Ада в язык структуризации текстов. Во-первых, это изменение порядка следования нетерминалов в правой части отдельного правила. Во-вторых, включение в ЯСС узкого класса фраз, которые на самом деле удовлетворяют правилам Ады, тогда как сами эти правила в общей форме к синтаксису разрабатываемого ЯСС не присоединяются. Технические модификации проекций, как например, переименование нетерминалов или упрощение избыточных ограничителей типа «end;», не упоминаем.

В результате создан и сейчас будет описан (в основном с позиций синтаксиса) язык, который именуем ЯСС первого уровня или, ЯСС версии 1.5.

3. Результат - синтаксис ЯСС версии 1.5

Данный язык предназначен для описания модульной структуры и межмодульных связей, которые могут быть усмотрены экспертом в текстовых материалах, размещаемых в базе данных информационной системы некоторого научного направления. С точки зрения этого языка смысл процесса размещения, далее именуемого просто размещением, состоит в передаче в базу данных системы упомянутых материалов в форме структурированных аннотаций, а затем (по мере появления возможностей) в форме реализаций - более или менее полных соответственно структурированных текстов. В реализациях, в частности, отмечаются места и виды смысловых связей с другими материалами, другими частями данного материала или с опубликованной литературой.

В начале каждого правила контекстно-свободного синтаксиса ЯСС даём его пояснение (включая важнейшие сведения о контекстных ограничениях), далее - само правило в расширенной форме Бэкуса-Наура, а в конце - соответствующее правило из формального синтаксиса языка Ада. При этом даётся ссылка на конкретный пункт стандарта языка Ада [5], из которого отбирались эти правила в процессе проектирования на вышеупомянутое множество понятий. Разумеется, в нашем изложении эти правила выглядят как «сокращения» правил из [5], поскольку в записи правил на основании расширенной формы Бэкуса-Наура содержится множество правил, имеющих стандартную форму. При проектировании такое множество, вообще говоря, сокращается.

При видимом расхождении между структурой правила ЯСС и структурой соответствующего ему правила из проекции языка Ада, тут же разъясняем, за счёт чего это противоречие преодолевается. Курсив в правилах отмечает незначимые (комментирующие) части в названиях синтаксических понятий.

В ЯСС лексика служебных терминалов (обязательно обрамляемых парами косых черт), нетерминалов (обязательно начинающихся с букв верхнего регистра, притом, что далее - только буквы нижнего регистра), а также идентификаторов ("названий") самого языка имеет особенность. В их состав могут входить пробелы, символы табуляции и концы строк, причём для идентификации важно только место вхождения, а не количество и вид таких разделителей.

Правило 1: Размещение это последовательное во времени размещение модулей. Порядок в этой последовательности, вообще говоря, существенен.

Размещение ::= { Размещаемый модуль }

Прообраз правила выбран из пункта (10.1.1.2) руководства по языку Ада [5]:
compilation ::= { compilation_unit }

Правило 2: Размещаемый модуль состоит из материала, предварённого предписанием контекста, которое, в частности определяет, какие размещаемые модули должны были быть размещены ранее данного.

Размещаемый модуль ::= Предписание контекста Материал

Соответствующее правило выбрано из (10.1.1.3) [5]:

compilation_unit ::= context_clause library_item

Правило 3: Под материалом понимается аннотация цельного по смыслу текстового материала или его реализация. Последняя есть некоторая версия этого текста (частичная, полная или полная комментированная).

Материал ::= Аннотация материала | Реализация материала

Из (10.1.1.4) [5]: library_item ::= library_unit_declaration | library_unit_body

Правило 4: Аннотация материала прежде всего вводит форму этого материала - он может быть аннотацией решения, контекста, аннотацией прототипа или же применением прототипа.

Аннотация материала ::= Аннотация решения | Аннотация контекста
| Аннотация прототипа | Создание по прототипу

Из (10.1.1.5) :
library_unit_declaration::= subprogram_declaration | package_declaration
| generic_declaration | generic_instantiation

Правило 5: Реализация материала размещается после его аннотации и имеет определяемую аннотацией форму, так что может быть реализацией решения или контекста.

Реализация материала ::= Реализация решения | Реализация контекста

Из (10.1.1.7): library_unit_body ::= subprogram_body | package_body

Правило 6: Предписание контекста может быть представлено (часто не единственным образом) последовательностью секций контекста. Порядок следования секций на смысл не влияет.

Предписание контекста::= { Секция контекста }

Из (10.1.2.2): context_clause ::= { context_item }

Правило 7: Секция контекста может служить предписанием источника (источников), т.е. указывать размещённые ранее модули, от которых зависит по смыслу данный модуль. Иначе секция контекста может предписывать возможность свободного использования терминологии каких-то из только что предписанных контекстов.

Секция контекста ::= Предписание источника | Предписание терминологии

Из (10.1.2.3) [5]: context_item ::= with_clause | use_clause

Правило 8: Предписание источника перечисляет названия материалов, на которые можно (и по смыслу - должно) ссылаться из данного материала. Если предписанный источник имеет форму решения, то можно указывать использование его самого. Если это контекст, то можно использовать всё, что видимо в данном контексте. Если предписанный источник - прототип, то по нему можно создавать новые решения или контексты в зависимости от того, прототипом чего он является. Если предписанный источник является потомственным материалом, то, наряду с ним, предписанными считаются и все его прямые предки.

Предписание источника ::= //в контексте// Название *материала*
{ //и// Название *материала* }

Из (10.1.2.4) [5]: with_clause ::= **with** *library_unit_* name { **,** *library_unit_* name }**;**

Правило 9: Предписание терминологии может относиться только к контекстам, видимым в месте расположения данного предписания.

Предписание терминологии ::= Предписание терминологии контекста

Из (8.4.2): use_clause ::= use_package_clause

Правило 10: Предписание терминологии контекста перечисляет контексты и до конца текущего модуля разрешает для случаев, когда это не приводит к противоречиям, свободное использование всего того, что видимо в предписанных контекстах.

Предписание терминологии контекста ::= //в терминах// Название *контекста*
{ //и// Название *контекста* } //;//

Из (8.4.3): use_package_clause ::= **use** *package*_name { **,** *package*_name }**;**

Правило 11: Аннотация решения представляет собой оглавление, ограниченное справа зарезервированным символом.

Аннотация решения ::= Оглавление решения //;//

Из (6.1.2): subprogram_declaration ::= subprogram_specification;

Правило 12: Для модуля материала, имеющего форму решение, оглавление вводит название и пояснение решения.

Оглавление решения ::= //результат// Вводимое название модуля материала
Пояснение решения//;//

Из (6.1.4):
subprogram_declaration ::= **procedure** defining_program_unit_name formal_part;

Правило 13: Аннотация контекста представляет собой оглавление контекста.

Аннотация контекста ::= Оглавление контекста

Из (7.1.2) [5]: package_declaration ::= package_specification;

Правило 14: Для модуля материала, имеющего форму контекста, оглавление вводит его название и последовательность основных элементов содержания, которые видимы отовсюду, где видим данный контекст.

Оглавление контекста ::=
//контекст// Вводимое название модуля материала //это//
{ Основной элемент содержания }
//всё//

Из (7.1.3): **package** defining_program_unit_name **is**
{ basic_declarative_item }
end ;

Правило 15: Для модуля материала, имеющего форму прототипа, аннотация может быть аннотацией метода решения или аннотацией схемы контекста.

Аннотация прототипа::= Аннотация метода решения
| Аннотация схемы контекста

Из (12.1.2):
generic_declaration ::= generic_subprogram_declaration
| generic_package_declaration

Правило 16: Аннотация метода решения это оглавление решения, предваренное пояснением применения прототипа, где: объясняется, что подлежит конкретизации при конкретном применении данного метода.

Аннотация метода решения ::= //заготовленный//
Пояснение к применению прототипа
Оглавление решения

Из (12.1.3): generic_subprogram_declaration ::=
generic [generic_formal_parameter_declaration] subprogram__declaration

Правило 17: Аннотация схемы контекста это оглавление контекста, предваренное пояснением применения прототипа, объясняющим, что подлежит конкретизации при конкретном воплощении данной схемы.

Аннотация схемы контекста ::=
//заготовленный// Пояснение к применению прототипа Оглавление контекста

Из (12.1.4): generic_package_declaration ::=
generic [generic_formal_parameter_declaration] package_ declaration

Правило 18: Реализация решения воспроизводит оглавление решения (по аннотации), вводит (до конца реализации) собственный контекст и сопровождает текст данного модуля гипертекстовыми ссылками на фрагменты, вынесенные в собственный контекст.

Реализация решения ::= Оглавление решения //это//
Собственный контекст
//начало//
Комментированное сопровождение текста //всё//

Из (6.3.2) [5]: subprogram_body ::= subprogram_specification **is**
declarative_part
begin handled_sequence_of_statements **end;**

Правило 19: Реализация контекста состоит в развёртывании этого (ранее аннотированного) контекста. Это означает, во-первых, что аннотированные в оглавлении контекста модули материала (из числа основных элементов содержания) получают свои реализации в собственном контексте этой реализации. Во-вторых, сопровождение текста содержит «служебную» часть текста данного модуля материала - то, что не вошло в структурированном виде в собственный контекст реализации.

Реализация контекста ::= //развёрнутый// //контекст//
Идентифицирующее название материала //это//
Собственный контекст
[//начало //
Последовательное комментированное изложение]
//всё//

Из (7.2.2): package_body ::=
package body defining_package_identifier **is**
declarative_part
[**begin** handled_sequence_of_statements] **end**;

Правило 20: Создание по прототипу это, во-первых, именование создаваемого решения или контекста, далее - указание по имени того метода решения или схемы контекста, которые послужили прототипом, и, наконец, - пояснение использования прототипа (которое, может отсутствовать, если в пояснении применения прототипа предусмотрен стандартный - умалчиваемый вариант использования)

Создание по прототипу ::=
//контекст// Вводимое название модуля материала //это//
//производный// Имя *схемы контекста*
[Пояснение использования прототипа]//;//
//результат// Вводимое название модуля материала //это//
//производный// Имя *метода решения*
[Пояснение использования прототипа]//;//

Из (12.3.2) [5]: generic_instantiation ::=
package defining_program_unit_name **is new** generic_package_name
[generic_actual_part];
| **procedure** defining_program_unit_name **is new** generic_procedure_name
[generic_actual_part];

Правило 21: Размещаемый модуль материала может представлять весь материал или его начальную часть или быть продолжением другого модуля (например, начальной части материала), который в этом случае должен иметь форму контекста и называется родительским модулем. У данного размещаемого модуля может быть сколько угодно прямых продолжений. Все они вместе со своими продолжениями (и продолжениями продолжений) называются потомками данного модуля. Родительский модуль видим в каждом из своих потомков, но не наоборот. Потомки одного поколения также не видят друг друга иначе, как с помощью предписаний контекста.

Вводимое название модуля материала ::=
[Название родительского модуля //:://] Вводимое название

Из (6.1.7):
defining_program_unit_name ::= [parent_unit_name .]defining_identifier

Правило 22: Название родительского модуля ::= Название

Из (10.1.1.8): parent_unit_name ::= name

Правило 23: Название ::= Непосредственое название | Именованная часть

Из (4.1.2): name ::= direct_name | selected_component

Правило 24: Непосредственное название ::= Идентификатор

Из (4.1.3): direct_name ::= identifier

Правило 25: Именованная часть::= Название части //из// Адресация

Из (4.1.3.2): selected_component ::= prefix . selector_name
Преодоление различия – изменение порядка «слева-направо» на порядок «справа-налево».

Правило 26: Название части::= Идентификатор

Из (4.1.3.2): selector_name ::= identifier

Правило 27: Адресация ::= Название

Из (4.1.4) [5]: prefix ::= name

Правило 28: Пояснение решения в аннотации решения, если такое пояснение есть, представляет собой пояснение применения, разъясняющее, из чего в этом решении исходим, что получаем.

Пояснение решения::= [Пояснение применения]

Из (6.1.12):
parameter_profile ::= [formal_part]

Правило 29: Пояснения применения могло бы состоять из нескольких разделов, но на данном уровне структуризации нет смысла выделять различные разделы в пояснениях к одному результату, поскольку детализация разделов не предусмотрена.

Пояснения применения ::= //:// Раздел пояснений

Из (6.1.14): formal_part ::= (parameter_specification)

Правило 30: На элементы текстового содержания данного модуля материала можно указывать из различных мест текста данного, а иногда и других модулей. В то же время средствами ЯСС невозможно никакое указание на элемент текста, не оформленный в качестве элемента содержания.

Собственый контекст ::= {Элемент содержания}

Из (3.11.2): declarative_part ::= {declarative_item}

Правило 31: Элементы содержания делятся на две группы. В одной из них те, "основные", для которых характерно то, что они связаны с именованиями, позволяющими указывать элементы текста по названиям. Во второй группе - реализации модулей, которые не определяют новых названий или доступа к ним

Элемент содержания ::= Основной элемент содержания | Реализация

Из (3.11.3): declarative_item ::= basic_declarative_item | body

Правило 32: Основные элементы содержания определяют новые термины для использования в последующих указаниях либо открывают потенциально прямой доступ к названиям ("терминам") из определённых контекстов

Основной элемент содержания ::=
Основное описание | Предписание терминологии

Из (3.11.4): basic_declarative_item ::= basic_declaration | use_clause

Правило 33: Основное описание может определять новое понятие, вводить название для порции информации, соответствующей известному понятию, именовать конкретные числовые значения, а также основное описание может быть аннотацией модуля материала или вводить модуль материала, определяемый прототипом.

Основное описание ::=
Определение | Элементы справочной информации | Именование чисел
| Аннотация решения | Аннотация контекста
| Аннотация прототипа | Создание по прототипу

Из (3.1.3) [5]:
basic_declaration ::= type_declaration | object_declaration | number_declaration
| subprogram_declaration | package_declaration
| generic_declaration | generic_instantiation

Правило 34: В данной версии только одна - обычная форма определения.

Определение ::= Обычное определение

Из (3.2.1.2):
type_declaration ::= full_type_declaration

Правило 35: Для определения понятия необходим термин и определение.

Обычное определение ::= //определение//
Определяемый термин [//это// Формулировка определения] //;//

Из Ада (3.2.1.3): full_type_declaration ::= **type** defining_identifier **is** type_definition;

Преодоление различия: «//определение// Определяемый идентификатор //;//» означает
//определение// Определяемый идентификатор //это// здесь //;//
и соответствует
type defining_identifier is new Here;
где Here обозначает личный тип (в смысле языка Ада), предопределённый в системе поддержки Ада-модулей, сопряжённых с модулями материалов.

Правило 36: Элементы справочной информации описывают такие количества, величины, перечни, таблицы и т.п., которые играют в текстах материалов особую роль. Этот вид основных описаний позволяет поименовать нужный информационный элемент при обязательном условии указания понятия, которое определяет характер предоставляемой этим элементом информации. В одном описании элементов справочной информации можно ввести перечень названий нескольких однотипных элементов. Можно также связать с определяемыми элементами выражение, которое будет их подразумеваемым значением.

Элементы справочной информации::= Перечень определяемых элементов //:// Характер информации [//есть// Выражение]

Из (3.3.1.2) [5]:
object_declaration ::= defining_identifier_list : subtype_indication [:= expression];

Правило 37: Перечень определяемых элементов ::=
Вводимое название {//и// Вводимое название }

Из (3.3.1.3):
defining_identifier_list ::= defining_identifier {, defining_identifier}

Правило 38: Характер информации указывается с помощью ранее определённого (или предопределённого) термина, причём, возможно, с указанием дополнительной специализации.

Характер информации ::= Термин [Специализация]

Из (3.3.1.2): subtype_indication ::= subtype_mark [constraint]

Правило 39: В данной версии ЯСС специализировать понятие можно только путем уточнения параметров его структуры. При этом параметрами структуры в данной версии могут обладать только предопределённые понятия (например, **библиогр** - информация об использованных литературных источниках).

Специализация ::= Специализация структуры

Из (3.2.2.5): constraint ::= composite_constraint

Правило 40: Параметры структуры, уточнение которых возможно, характеризуют регулярные перечни (списки, таблицы) или же признаки нерегулярных структур. В синтаксисе ЯСС первого уровня нет средств явного определения понятий в форме перечней или в форме с уточняемыми признаками. Но такие понятия предопределяются в ЯСС или системой поддержки ЯСС. Например, предопределённый термин **библиогр** требует уточнения диапазона номеров цитируемых источников.

Специализация структуры ::= Уточнение перечней | Уточнение признаков

Из (3.2.2.7): composite_constraint ::= index_constraint | discriminant_constraint

Правило 41: Форма уточнения регулярного перечня (массива, таблицы) в данной версии ЯСС предопределяется только в каждом конкретно случае вместе с предопределяемым термином. Естественно, будет требоваться указывать диапазоны индексов, например, **библиогр (1..12)** .

Уточнение перечней ::= //(//Чёткий диапазон {//и// Чёткий диапазон }//)//

Из (3.6.1.2) [5]: index_constraint ::= (discrete_range { , discrete_range })

Правило 42: Форма уточнения признаков для предопределённого термина должна быть также для него предопределена. Конкретизация признака в данной версии ЯСС формально не детализируются, но по смыслу это должно быть выражение подходящего типа. Например, **табл (7, число клиентов)**.

Уточнение признаков ::=
//(// Конкретизация признака { //и// Конкретизация признака } //)//

Из (3.7.1.2): discriminant_constraint ::=
(discriminant_association { , discriminant_association })

Правило 43: Если текст материала вводит числовое значение, имея в виду на него ссылаться, то это введение следует оформить как именование числа. Последнее может вводить не одно, а несколько разных названий числового значения. Само значение должно до конца вычисляться из выражения, форму которого данная версия ЯСС не ограничивает.

Именование чисел ::= Перечень определяемых элементов //:://
//всегда// //есть// Выражение, *вычислимое до конца* //;//

Из (3.3.2.2): number_declaration ::=
defining_identifier_list **: constant :=** *static*_expression**;**

Правило 44: В данной версии ЯСС сопровождение текста обходится без комментаириев.

Комментированное сопровождение текста ::= сопровождение текста текста

Из (11.2.2): handled_sequence_of_statements ::= sequence_of_statements

Правило 45: Сопровождение текста может включать указания, выявляющие наиболее общие и, наиболее важные информационные связи в рамках и за рамками данного материала.

Сопровождение текста ::= Указание { Указание }

Из (5.1.2): sequence_of_statements ::= statement { statement }

Правило 46: В данной версии ЯСС все указания являются прямыми.

Указание ::= Прямое указание

Из (5.1.3): statement ::= simple_statement

Правило 47: Параметры справочной информации указываются в форме значений размеров и элементов перечней, таблиц и т.п., а указания на использование модулей-результатов раскрывают систему смысловых зависимостей в системе модулей размещённых материалов.

Прямое указание ::=
Указание значения | Указание использования результатов

Из (5.1.4) [5]: simple_statement ::= assignment_statement | procedure_call_statement

Правило 48: Перечни, элементы таблиц и т.п. могут получать, как числовые, так и описываемые словами значения там, где они видимы как элементы справочной информации

Указание значения ::=
Название *элемента справочной информации* //есть// Выражение //;//

Из Ада (5.2.2): assignment_statement ::= *variable*_name := expression;

Правило 49: Название результата, с которого начинается конструкция указание использования результата, имеет форму, зависящую от места, где эта конструкция находится. Если аннотация используемого результата видна с этого места прямо (находится в том же модуле или корректно подпадает под предписание терминологии объемлющего аннотацию контекста), то достаточно использовать непосредственное название результата, т.е. его идентифицирующее название, определённое в аннотации. Если же видимость есть, но не прямая, то указывать использование результата можно, но его название должно иметь форму составного имени. Оно начинается с непосредственного название результата и далее через разделители //из// - названия контекстов, объемлющих аннотацию данного результата, начиная с непосредственно объемлющего и заканчивая наиболее внешним из тех, которые видимы в месте использования. Если этот наиболее внешний контекст является размещаемым модулем, но его название в свою очередь может включать названия родительских модулей (через разделитель //:///).

Указание использования результата ::=
Адресация *результата* [Пояснение использования] //;//

Из (6.4.2):procedure_call_statement ::= *procedure*_name;
| *procedure*_prefix actual_parameter_part;

Правило 50: Если пояснение решения в оглавлении решения было не пустым, а имело пояснение применения, то указание использования данного результата должно иметь соответствующее пояснение использования.

Пояснения использования ::= //(// Соответствующее пояснение //)//

Из (6.4.4) [5]: actual_parameter_part ::= (parameter_association)

Правило 51: Следующие нетерминалы замещаются текстом на естественном языке в соответствии со смыслом проводимой структуризации соответствующего модуля материала:

Вводимое название, Выражение, Идентификатор, Конкретизация признака, Пояснение использования прототипа, Пояснение к применению прототипа, Соответствующее пояснение, Раздел пояснений, Термин.

4. Заключение

Исследован вопрос о заимствовании из языка-прототипа правил контекстной зависимости при разработке синтаксиса языков разметки текстов на базе известного языка. В частности указаны условия корректности дополнения проекции контекстно-свободного синтаксиса соответствующей «проекцией» контекстных правил языка-прототипа.

Предложен метод для построения полезных в приложениях расширений «минимального» языка разметки, непосредственно определённого проекцией в понимаемом выше смысле.

Реализовано построение полного синтаксиса языка семантической структуризации (ЯСС) в версии 1.5, которая позволяет обслуживать все стадии процесса размещения материалов в базе данных информационной системы научного направления.

Использование в качестве языка-прототипа известного языка программирования Ада обеспечивает возможность практического внедрения нового языка ЯСС 1.5 путём подключения стандартных информационных технологий, связанных с языком Ада.

ЛИТЕРАТУРА

1. Мищенко В.О. Развивающаяся информационная система научного направления с выходом программного обеспечения // Інформаційна інфраструктура вищих закладів освіти / Зб.наук. праць. Том 2 . - Херсон: Херсонський державний педагогічний ун-т, 2000. - С. 208-216.
2. Мищенко В.О. Семантическая структуризация текстовых данных в базе данных, поддерживающей распределенную разработку программного обеспечения // Вестник ХГТУ. – 2002, № 1 (14). – С. 304-307.
3. Кауфман В.Ш. Языки программирования. Концепции и принципы. – М.: Радио и связь, 1993. – 432 с.
4. Ахо А., Ульман Дж. Теория синтаксического анализа, перевода и компиляции. Том 1: Синтаксический анализ. – М.: Мир, 1978. – 497 с.
5. Taft S. Tucker,Duff Robert A.(eds) Ada Reference Manual: Language and Standard Libraries, International Standard ISO/IEC 8652:1995(E).-Lecture Notes in Computer Science.- vol.1246.- Springer-Verlag, 1997, ISBN 3-540-63144-5.