OSTIS-2011
(Open Semantic Technologies for Intelligent Systems)

УДК 004.822:43

ТЕХНОЛОГИЯ ПРОЕКТИРОВАНИЯ ПРОГРАММ, ОРИЕНТИРОВАННЫХ НА ОБРАБОТКУ СЕМАНТИЧЕСКИХ СЕТЕЙ

Д.А. Лазуркин (*dilaz03@gmail.com*)
Белорусский государственный университет информатики и радиоэлектроники, г.Минск, Республика Беларусь

В работе приводится описание основных разделов технологии проектирования программ, ориентированных на обработку семантических сетей. Результаты, описанные в данной работе, являются частью международного открытого проекта OSTIS (Open Semantic Technologies for Intelligent Systems). Использование предложенной технологии позволяет эффективно разрабатывать программы для интеллектуальных систем.

Ключевые слова: абстрактная машина обработки информации, отладка, программа, семантическая сеть, язык программирования

Введение

Перед разработчиками программных систем встают все более сложные задачи: программные системы должны решать огромное количество классов задач. Это требует их большей интеллектуализации. Сложность такого программного обеспечения возрастает, следовательно, необходим переход к программным системам иного качества. Поэтому на замену традиционным системам приходят интеллектуальные системы, т.е. системы, в основе которых лежит формальное представление используемых ими знаний.

Реализация перехода от традиционных систем к интеллектуальным системам становится возможной при упрощении процесса разработки последних, что, в свою очередь, влечет необходимость создания технологии, ориентированной на системы такого класса. Задача создания такой технологии решается в рамках международного открытого проекта «Открытая компонентная семантическая технология для интеллектуальных систем». Эта технологии является комплексной и состоит из более частных технологий. В данной статье будет рассмотрена одна из частных технологий, которая ориентирована на проектирование программ, ориентированных на обработку семантических сетей. В ее состав входит:

- теория языков программирования и программ, ориентированных на обработку семантических сетей;
- библиотека совместимых ip-компонентов программ, ориентированных на обработку семантических сетей;
- инструментальные средства проектирования программ, ориентированных на обработку семантических сетей;
- методика проектирования программ, ориентированных на обработку семантических сетей;
- методика обучения проектированию программ, ориентированных на обработку семантических сетей;
- интеллектуальная help-система по технологии проектирования программ, ориентированных на обработку семантических сетей.

В данной статье будут затронуты только вопросы, связанные с созданием теории языков программирования, ориентированных на обработку семантических сетей, и поддержкой их

реализации на различных платформах. Также будут рассмотрены вопросы создания интегрированного средства отладки программ, написанных на таких языках программирования.

1. Языки программирования, ориентированные на обработку семантических сетей

Каждому языку программирования можно поставить в соответствие его синтаксис, отражающий свойства внутреннего строения конструкций этого языка, и денотационную семантику, отражающую общие свойства соотношения между конструкциями языка и описываемой ими предметной областью. Однако для языка программирования более простым и интуитивно понятным является описание операционной, а не денотационной семантики. Т.е. описание правил манипулирования конструкциями языка программирования, направленные на решение различных задач в рамках этого языка [**Непейвода, 1983**].

Для формального задания формального задания операционной семантики языка программирования в работе [**Голенков и др, 2001а**] вводится понятие абстрактной машины обработки информации, которая состоит из:

- некоторым образом организованной памяти (запоминающая среда), в которой хранятся перерабатываемые информационные конструкции;
- множества операций (правил вывода), выполняемых над указанной памятью.

Программы операций такой абстрактной машины записываются на языке микропрограммирования и называются микропрограммами. Микропрограммы не должны непосредственно храниться в памяти этой абстрактной машины. Нас будут интересовать абстрактные машины, которые имеют графовую ассоциативную структурно перестраиваемую память, т.е. графодинамические абстрактные машины. За счет того, что память таких машин является графовой, она очень хорошо подходит для хранения семантических сетей. Процесс обработки семантической сетей в такой памяти является графодинамическим процессом, т.е. процессом, который изменяет конфигурацию семантической сети. За счет свойств перестраиваемости структуры памяти и ассоциативности доступа к ней обработка семантических сетей является простой и эффективной.

В рамках проекта OSTIS [**OSTIS, 2011**] для описания и реализации графодинамических моделей обработки информации предлагается семейство абстрактных sc-машин и семейство соответствующих им sc-языков программирования. Особенностью данного семейства абстрактных машин является то, что все они основаны на SC-коде и используют sc-память для хранения и переработки информационных конструкций. Языки программирования, соответствующие данному семейству абстрактных машин, так же основаны на SC-коде. Это значит, что программы на таких языках имеют вид sc-конструкций и хранятся в sc-памяти. Это позволяет обрабатывать программы также как и любые другие информационные конструкции. Это свойство является важным подспорьем к созданию интеллектуальных программ.

Во главе семейства sc-языков программирования стоит язык процедурного программирования SCP (Semantic Code Programming). Ему отводится важная роль, потому что абстрактная scp-машина является базовым уровнем интерпретации других абстрактных sc-машин. Т.е. микропрограмма операции любой sc-машины, кроме scp-машины, может быть реализована на языке SCP. Это позволяет обеспечивать высокую степень независимости от конкретных реализаций sc-машин на программных или аппаратных платформах. Синтаксис и денотационная семантика каждого scp-оператора описаны в подпроекте 150 проекта OSTIS. Абстрактная scp-машина описывается в рамках подпроекта 151, но нее описание еще не завершено.

Язык SCP является низкоуровневым языком процедурного программирования, поэтому для более эффективного решения задач необходимо создание высокоуровневых языков программирования. На такую роль позиционируется язык SCPH (Semantic Code Programming Highlevel), при помощи которого на уровне операционной семантики можно манипулировать sc-конструкциями любых размеров. Описание данного языка приведено в подпроекте 400 проекта OSTIS. Так же для решения задач необходимо создание языков программирования,

которые используют непроцедурные подходы. Таким образом можно привести следующее, конечно неполное, семейство sc-языков программирования:

- процедурные sc-языки программирования:
  - язык SCP – базовый язык программирования ассемблерного типа;
  - язык SCPH – наследник языка SCP с более высокоуровневой операционный семантикой;
  - функциональный язык программирования.
- непроцедурные sc-языки программирования:
  - логические языки программирования на базе языка SCL;
  - продукционные языки программирования на базе языка SCL;
  - язык вычисления арифметических выражений.

Рассмотрим более подробно sc-языки процедурного программирования и способы задания состояний соответствующих им абстрактных sc-машин. При создании семейства таких языков целесообразно как можно в большей мере унифицировать их синтаксис. Это обеспечит существенное упрощения при интеграции программ на этих языках и их интерпретации. Так во всех языках процедурного программирования, построенных на базе SC-кода, некоторую программу ***pi*** (рисунок 1) можно трактовать как кортеж, компонентами которого являются:

- sc-узлы, обозначающие операторы. Операторы также являются кортежами, компоненты которых называются операндами. Операнды упорядочены атрибутами ***1_***, ***2_***, ***3_*** и т. д. На рисунок 1 из этих операторов приведен только оператор, с которого начинается выполнение программы;
- sc-узел, обозначающий множество всех фиксированных для данной программы sc-конструкций (обозначим его ***pci***);
- sc-узел, обозначающий множество всех программных переменных данной программы (обозначим его ***pvi***). Значением переменной программы ***pi*** является sc-элемент, связанный бинарным ориентированным отношением с sc-узлом, обозначающим переменную;
- sc-узел, обозначающий множество входных и выходных параметров данной программы (обозначим его ***ppi***). Это множество упорядочено атрибутами ***1_***, ***2_***, ***3_*** и т. д. Параметры, значения которых программа получает из вызвавшей ее программы, уточняются атрибутом ***in_***. Параметры, значения которых программа возвращает вызвавшей ее программе, уточняются атрибутом ***out_***;
- sc-узел, обозначающий оператор, с которого начинается выполнение данной программы (обозначим его ***opi***).

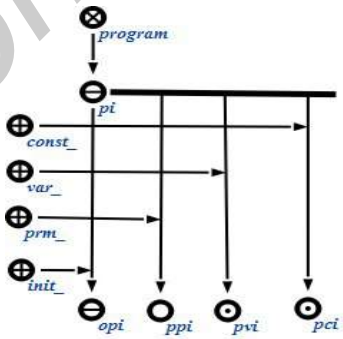

Рисунок 1 – Описание программы ***pi***

Для передачи потока управления от оператора к оператору используются дополнительных операндов. По характеру передачи управления все операторы процедурного языка можно разделит на три категории:

- операторы, при реализации которых осуществляется безусловная передача управления (рисунок 2);

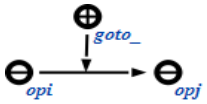

Рисунок 2 – Безусловный переход

- операторы, при реализации которых осуществляется условная передача управления одному из двух операторов (рисунок 3). При истинном условии передача управления оператору ***opj***, при ложном – оператору ***opk***;

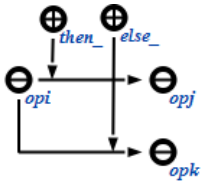

Рисунок 3 – Условный переход

- операторы, при реализации которых осуществляется завершение программы и формирование возвращаемых значений.

Состояние процесса интерпретации программы абстрактной sc-машины, которая соответствует описанному процедурному языку программирования, задается кортежем ***qi*** (рисунок 4), компонентами которого являются:

- sc-узел, обозначающий активный исполняемый оператор (opi);
- sc-узел, обозначающий родительский процесс для данного процесса (qj);
- sc-узел, обозначающий оператор родительского процесса, в результате интерпретации которого был порожден данный процесс (opj);

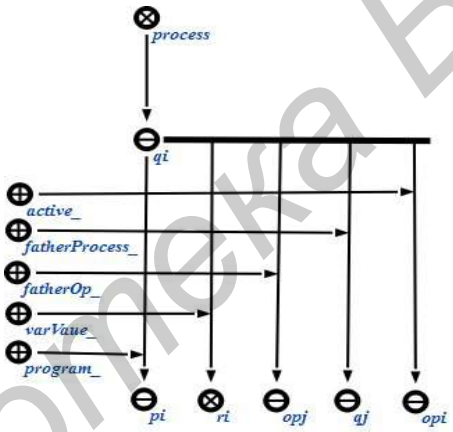

Рисунок 4 – Состояние процесса sc-машины

- sc-узел, обозначающий отношение текущих связей программных переменных(ri);
- sc-узел, обозначающий интерпретируемую программу (pi);

Кроме этого состояние процесса задается включением его знака в одно из следующих множеств:

- множество процессов, находящихся в состоянии перехода по then-пути выполнения. Знак процесса включается в данное множество, когда необходимо, чтобы следующим активным оператором стал оператор, знак которого является компонент с атрибутом then_ активного оператора;
- множество процессов, находящихся в состоянии перехода по else-пути выполнения. Знак процесса включается в данное множество, когда необходимо, чтобы следующим активным оператором стал оператор, знак которого является компонент с атрибутом else_ активного оператора;
- множество процессов, находящихся в состоянии повторения интерпретации предыдущего оператора;
- множество процессов, находящихся в состоянии приостановки и ожидания. Знак процесса включается в данное множество, когда необходимо приостановить выполнение этого процесса.
- множество процессов, находящихся в состоянии выполнения. Знак процесса включается в данное множество, когда процесс находиться в состоянии выполнения и следующим

активным оператором будет оператор, знак которого является компонентом с атрибутом ***goto_*** текущего активного оператора.

- множество процессов, находящихся в состоянии ошибки. Знак процесса включается в данное множество, когда в результате интерпретации активного оператора случается ошибочная ситуация.
- множество процессов, находящихся в состоянии завершения. Знак процесса включается в данное множество, либо в результате корректного завершения, либо в результате ошибки, когда данный процесс закончил свое существование.

2. Реализация языков программирования, ориентированных на обработку семантических сетей

В предыдущем разделе было отмечено, что базовым уровнем, который обеспечивает интерпретацию семейства sc-языков программирования, является абстрактная scp-машина. Значит для того чтобы обеспечить среду исполнения создаваемых программ на любых программных и аппаратных платформах необходима реализация этого уровня абстракции. Необходима реализация следующих основных компонентов:

- модель sc-памяти;
- набор операций, которые составляют scp-интерпретатор.

Явное выделение уровня абстракции среды исполнения позволяет обеспечить переносимость создаваемых программ, что в перспективе может привести к созданию специального графодинамического компьютера **[Голенков, 1994].**

На данном этапе развития компьютерной техники широко используются традиционные компьютеры с линейной организацией памяти. На таких компьютерах требуется специальная реализация модели sc-памяти. Важным моментом такой реализации является выбор правильной формы представления sc-конструкций в линейной памяти. Так как sc-конструкция является графом определенного вида, то для ее представления в линейной памяти подходит любая из форм кодирования графов. Однако, на деле, из классических форм кодирования графов подходят только адаптированные под задачу списки смежности и инцидентности, так как структура sc-памяти часто меняется, поэтому форма кодирования графов должна поддерживать быстрое добавление/удаление элементов и их компактное хранение. С целью поддержки механизмов защиты данных и структурирования пространства идентификаторов, а также улучшения эффективности поисковых операций, стандартная модель sc-памяти в реализации расширена до сегментной модели.

В сегментной модели вся память разбивается на сегменты, где каждый sc-элемент принадлежит ровно одному сегменту. Права любого пользователя на доступ к элементу равны его правам на доступ к сегменту. Однако в текущей версии библиотеки механизм защиты информации не реализован. Каждый сегмент имеет уникальное имя, и уникальность идентификаторов имеет место только в рамках сегмента. Таким образом, полное имя элемента состоит из имени сегмента и его идентификатора. Т.е. сегмент можно рассматривать как пространство имен. С учетом механизма защиты сегмент является некоторой логически обособленной частью базы знаний. Каждый сегмент имеет свой знак – узел, который находится в специальном сегменте – метасегменте. Именно в рамках метасегмента предполагается хранить информацию о правах доступа. Предполагается, что имя сегмента будет соответствовать схеме URI (Universal Resource Identifier). В текущей версии структура имени сегмента похожа на структуру пути на Unix-подобных файловых системах. Т.е. оба варианта предполагают древовидную структуру пространства имен сегментов.

Разные сегменты ничего не знают друг о друге. Таким образом, нужен способ сослаться из одного сегмента на элемент другого сегмента. Например, sc-дуга принадлежащая одному сегменту должна сослаться на ее конец, если он принадлежит другому сегменту. Для таких межсегментных связей был введен специальный элемент реализации – реализационная ссылка.

Для адресации элементов в sc-памяти в данной реализации введено понятие реализационного адреса. Реализационный адрес - это просто число. Ядро системы никак не трансформирует это

число. Реализационный адрес однозначно и уникально идентифицирует элемент заданного сегмента. Естественно, представление должно по возможности как можно быстрее переходить от реализационного адреса к кодирующим заданный элемент структурам. Для внутренних представлений логично использовать физический адрес таких структур в качестве реализационного адреса. Для того, чтобы полностью идентифицировать некоторый элемент необходимо указать его сегмент и его реализационный адрес в этом сегменте. Одному sc-элементу может соответствовать много пар "сегмент – реализационный адрес", т.к. ссылки тоже имеют реализационные адреса. Чтобы уникально и однозначно идентифицировать любой sc-элемент было введено понятие sc-адреса. В реализации sc-адрес это физический адрес небольшой структуры которая описывает сегмент и реализационный адрес элемента, является заголовком списка пар "сегмент – реализационный адрес" всех известных ссылок на элемент и содержит еще немного служебной информации.

Дерево сегментов и их содержимое хранится на файловой системе. Для представления древовидной структуры сегментов используются один в один иерархия папок и файлов. Каждый файл в такой иерархии является сегментом и имеет двоичный формат TGF (Transfer Graph Format), удобный для хранения и транспортировки sc-конструкций. Общая структура файла формата TGF состоит из заголовка и последовательности команд, каждая из которых формирует часть графа. Основными командами TGF являются:

- GenEl – генерировать элемент;
- SetBeg – установить начало дуги;
- SetEnd – установить конец дуги;
- DeclareSegment – объявляет сегмент с полным URI;
- SwitchToSegment – переключение на генерацию в сегмент;
- EndOfStream – определяет окончание передаваемого потока.

Последовательность инструкций в таком формате важна, так как команды получают свои аргументы, не только за счет явного их указания, но и за счет указания номера команды, результат которой необходимо использовать в качестве аргумента. Присутствие команд для работы с сегментами необходимо для того, чтобы использовать TGF не только для хранения, но и для пересылки данных по сети.

В таком бинарном виде хранятся sc-конструкции на файловой системе. Однако, при разработке баз знаний, частью которых могут являться программы, человек имеет дело с текстовыми или графическими данными, которые хранятся в репозитории исходных текстов.

На данный момент развития технологии программист использует текстовые форматы SCs и M4SCP для текстового кодирования sc-конструкций и написания scp-программ.

Для реализации интерпретатора scp-программ необходима среда выполнения и планирования процессов и операций обработки sc-памяти. Основную сложность здесь составляет обеспечение эффективности инициирования операций, так как они обладают условием активации, т. е. активируется только при появлении в sc-памяти определенной задачной sc-конструкции. Постоянный поиск таких конструкций является неэффективным, поэтому реализация sc-памяти должна поддерживать подпись обработчиков на различные происходящие в ней события. Такие события могут быть как простыми (генерация sc-элемента, удаление sc-элемента, включение sc-элемента в множество), так и более сложными (генерация в sc-памяти произвольной sc-конструкции). Однако в реализованной версии scp-интерпретатора исполнение программы происходит в виде последовательного исполнения каждого оператора без распараллеливания.

Текущая модель sc-памяти и scp-интерпретатора реализованы на языке C++ и стабильно работают на 32-х битной платформе Windows, а так же идет портирование на 32-ый битный Linux. Благодаря тому, что они реализованы как динамические библиотеки есть возможность создания к ним привязок для других языков программирования. Например, существуют привязки для языка Python.

3. Интегрированные средства отладки программ, ориентированных на обработку семантических сетей

Большую часть времени, затрачиваемую на разработку программного обеспечения, занимает поиск и исправление ошибок. Важное место в этом процессе отводится отладчику – средству, которое позволяет увидеть состояние программы во время выполнения. В средах программирования наличие данного программного обеспечения является стандартом де-факто.

В инструментальное средство проектирования программ, ориентированных на обработку семантических сетей, входит не только отладчик, но и редактор программ, сборщик репозитория исходных текстов, средство запуска программ, но в данной статье будет рассмотрено только средство отладки.

3.1. Традиционный подход к организации отладки программ

Стандартный цикл отладки начинается с того, что отладчик создает отлаживаемый процесс либо подключается к уже существующему процессу. В конце отладочной сессии отладчик либо завершает отлаживаемый процесс, либо просто отключается от него. Во время своей работы отладчик имеет доступ к таблице символьной информации, которая обычно хранится в файле вместе с исполняемым кодом. Данная таблица символьной информации позволяет совершать переход от инструкций в исходном тексте программ к адресам исполняемых инструкций в памяти. Эта информация позволяет отладчику устанавливать точки останова (ТО). Так же таблица символьной информации хранит информацию о программных переменных, так что отладчик может сопоставить переменным и структурам данных в исходном тексте места в памяти отлаживаемого процесса **[Rosenberg, 1996].**

3.2. Специфика отладки программ

Так как SCP является процедурным языком, то основная схема традиционного подхода отладки программ подходит и для создания отладчика scp-программ. Однако следует отметить, что тексты scp-программ представлены в виде семантических сетей и хранятся в ассоциативной памяти, что расширяет возможности отладки таких программ и упрощает сам отладчик по следующим причинам:

- структуру программ, представленных в виде семантических сетей, легче модифицировать, чем их линейные аналоги;
- ассоциативный доступ к памяти и явное выделение связей между частями scp-программы позволяет отладчику проявлять «интеллектуальность» без анализа низкоуровневой структуры отлаживаемой программы;
- часть символьной информации (имена программ, имена переменных) хранится в sc-памяти в виде идентификаторов соответствующих элементов;
- упрощается возможность добавления к программе символьной метаинформации.

3.3. Механизм отладки scp-программ

Для добавления в scp-интерпретатор возможности отладки scp-программ необходимо расширить модель scp-процесса. Процесс, интерпретирующий scp-программу, должен сообщать отладчику о ходе своей работы. Для этого введем понятие трассировщика scp-процесса. Трассировщик scp-процесса – это микропрограмма, которая сопоставлена данному scp-процессу и получает сведения о ходе его выполнения. Каждому scp-процессу может быть поставлен в соответствие один трассировщик. Таким образом, в связку, обозначающую scp-процесс, будет добавлен дополнительный компонент с атрибутом ptrace_, который является знаком трассировщика. Управление трассировщику передается в следующих случаях:

- осуществилось выполнение одного scp-оператора (только в случае пошаговой трассировки);
- в результате интерпретации scp-оператора произошла ошибка;
- scp-процесс создал процесс-потомок;
- scp-процесс завершился.

Для отладки scp-программ создана микропрограмма специального трассировщика, которая выполняет всю низкоуровневую часть отладки. Она имеет доступ ко всем структурам scp-

процесса и при необходимости может приостановить scp-процесс, переведя его в состояние сна. Эта же микропрограмма получает следующие сообщения управления от пользовательского интерфейса:

- продолжить выполнение текущего отлаживаемого scp-процесса, который приостановлен;
- включить режим пошаговой трассировки и перейти к следующему scp-оператору текущей scp-программы;
- перейти к начальному scp-оператору scp-программы, которую вызывает текущий scp-оператор.

При создании дочернего scp-процесса для него устанавливается трассировщик родительского scp-процесса, если родительский scp-процесс находится в режиме трассировки.

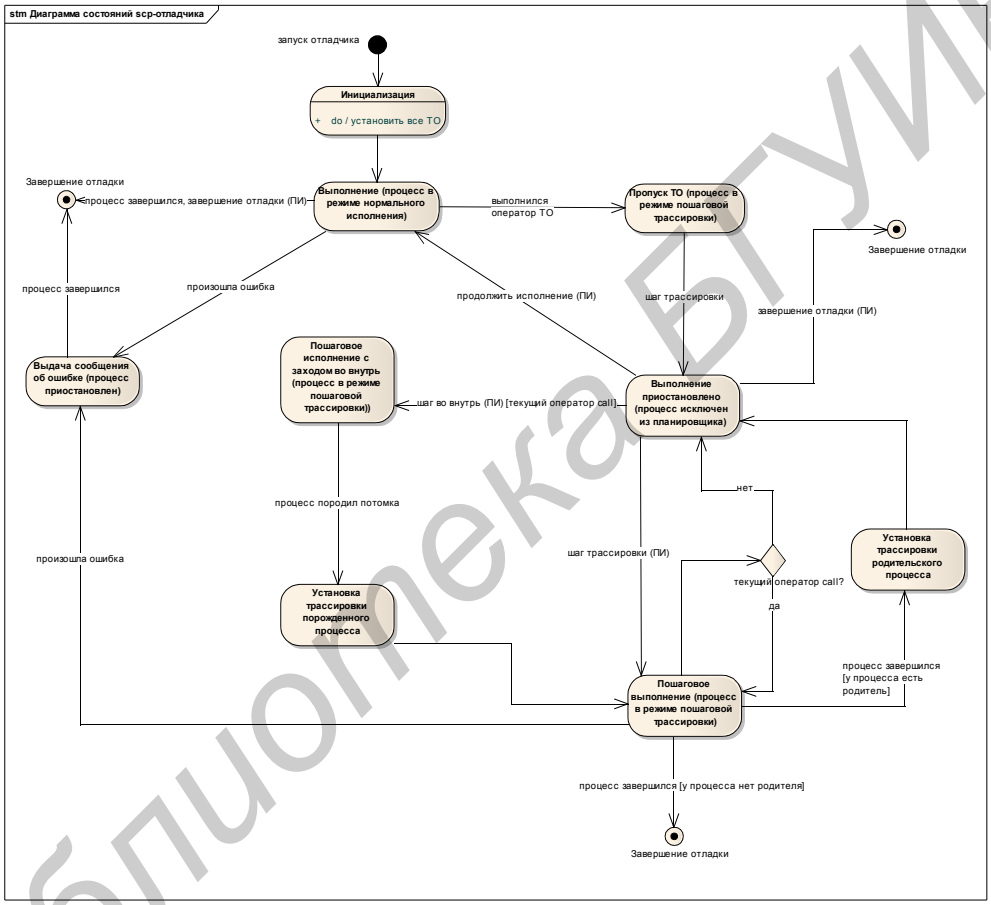

Рисунок 5 – Диаграмма состояний алгоритма работы scp-отладчика

3.4. Реализация точки останова

Традиционно для реализации функциональности точки останова используется специальная инструкция, при интерпретации которой управление передается отладчику для данного процесса, если он для этого процесса установлен.

Такая реализация точки останова может быть гибко поддержана в scp-интерпретаторе. Для этого введем дополнительный тип scp-оператора, который обеспечивает функциональность точки останова. У данного типа scp-оператора отсутствуют операнды. При передаче управления scp-оператору этого типа, интерпретирующая его операция находит знак трассировщика для текущего scp-процесса и передает микропрограмме трассировщика управление с указанием, что была активирована точка останова. Если трассировщик для данного scp-процесса не установлен, то этот процесс будет завершен.

Внедрение точки останова происходит путем модификации текста scp-программы, уже загруженного в ассоциативную память. Отладчик находит scp-оператор, на который необходимо установить точку останова по первому номеру строки оператора в исходном файле.

Перед установкой точки останова scp-отладчик копирует программу во избежание конфликтов с другими scp-процессами, которые могут интерпретировать эту программу. На рисунке 6 приведен пример установки scp-оператора ***brk*** типа точка останова на scp-оператор ***opj***, к которому только один безусловный переход от scp-оператора ***opi***. Алгоритм такой установки очень прост и основан на ассоциативном поиске всех входящих дуг интерпретируемой scp-программы, которые определяют передачу управления, в scp-оператор, на который устанавливается точка останова. Для всех найденных дуг их конечным элементов устанавливается знак созданной точки останова, а от самой точки останова к scp-оператору, на который она установлена, создается безусловный переход.

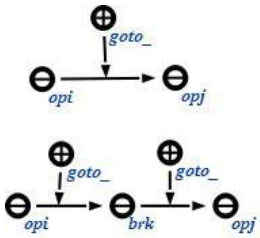

Рисунок 6 – Способ установки точки останова на scp-оператор ***opj***

3.5. Визуальное отображение процесса отладки scp-программ

Для полноты функциональности scp-отладчика необходимо, что бы он мог отображать значения scp-переменных в процессе исполнения. Это возможно при помощи компонента ядра семантического пользовательского интерфейса. Данный компонент обеспечивает для отладчика отображение фрагментов семантических сетей с их оптимальным размещением в окне, что позволяет визуализировать графодинамический процесс преобразования памяти.

Основной информацией, с которой работает отладчик для визуализации памяти, являются значения scp-переменных. Значения scp-переменных являются опорными точками, от которых отладчик может отображать содержимое памяти.

На рисунке 7 показан способ задания для scp-переменной ***vari*** в качестве значения узла ***vali***. В таком способе задания можно выделить следующие структурные элементы:

- заголовок scp-переменной для отображения на экране – это элементы с идентификаторами vari и valpair;
- собственно само значение vali.

На экран пользователю достаточно выводить только эту информацию. Причем значение scp-переменной выводится с некоторой глубиной поиска по ассоциативным связям, которую задает разработчик программ на языке SCP, когда хочет просмотреть значение переменной.

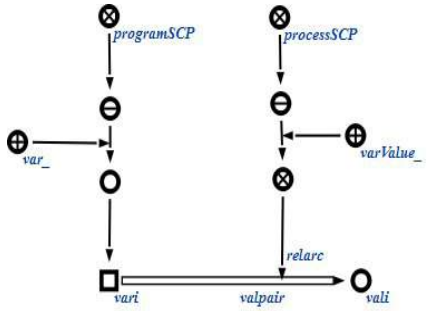

Рисунок 7 – Способ задания для scp-переменной ***vari*** значения ***vali***

С учетом специфики отладки scp-программ при отображении значений scp-переменных возникают следующие проблемы:

- для удобства отладки необходимо обеспечить оптимальное размещение конструкций семантических сетей на экране так, чтобы разработчик мог легко увидеть изменение, которое произошло в выводимом участке памяти при выполнении одного шага отладки;
- неизвестен размер семантической окрестности, доступной по ассоциативным связям, значения scp-переменной, которую необходимо вывести на экран;
- неизвестны элементы, которые необходимо удалить с экрана, при выключении отображения значения scp-переменной (кроме тривиальных случаев);
- в некоторых случаях количество элементов, находящихся в непосредственной семантической близости от значения выводимой scp-переменной очень велико, что препятствует.

Заключение

В настоящее время ведется работа по реализации описанной технологии проектирования программ. Уже на данном этапе уже можно сказать, что она позволяет быстро проектировать программы на языке SCP с использованием интегрированного инструментального средства. Основным недостатком, который присутствует на данный момент, является отсутствие библиотеки ip-компонентов программ, которая может еще более ускорить и упростить процесс проектирования. Дальнейшее развитие направлено на создание библиотеки ip-компонентов, более четкую разработку методики проектирования программ и создание help-системы, что в совокупности обеспечит массовость технологии, а также сокращение сроков разработки программ.

Работа поддержана грантом БРФФИ-РФФИ Ф10Р-175.

Библиографический список

[Голенков, 1994] Голенков, В.В. Параллельный графовый компьютер (PGC), ориентированный на решение задач искусственного интеллекта, и его применение / В.В. Голенков. – Минск, 1994. – 60 с. – (Препринт / Ин-т техн. Кибернетики АН Беларуси; № 2).

[Голенков и др, 2001a] Представление и обработка знаний в графодинамических ассоциативных машинах /В. В. Голенков, [и др]; – Мн. : БГУИР, 2001. – С.52-55.

[Голенков и др, 2001b] Программирование в ассоциативных машинах / В.В. Голенков [и др.]. – Минск, БГУИР, 2001 – 276 с.

[Касьянов и др, 2003] Касьянов, В.Н. Графы в программировании: обработка, визуализация и применение/ Касьянов В.Н., Евстигнеев В.А. – С.-П.: BHV, 2003 – 1104 с.

[Кузнецов, 1989] Кузнецов, В.Е. Представление в ЭВМ неформальных процедур: продукционные системы / Кузнецов В.Е. – М.: Наука, 1989 – 160 с.

[Левин, 1983] Левин, Д.Я. Язык сверхвысокого уровня СЕТЛ и его реализация (для ЭВМ БЭСМ–6) / Левин Д.Я. – Новосибирск: Наука, 1983 – 158 с.

[Непейвода, 1983] Непейвода, Н.Н. Семантика алгоритмических языков // Итоги науки и техники. - Сер.: Теория вероятностей. Математическая статистика. Теоретическая кибернетика. - М.: ВИ- НИТИ АН СССР, 1983. - Т. 20. - с. 95-166.

[Тыугу, 1984] Тыугу, Э.Х. Концептуальное программирование/ Э.Х. Тыугу. – М.: Наука, 1984. – 256 с.

[Уварова и др, 1987] Уварова, Т.Г. Формальное описание операционного языка для семантических сетей / Уварова Т.Г., Лифшиц Л.Л. - М.: ВЦ АН СССР, 1987. – 33 с.

[Успенский и др, 1987] Успенский, В.А. Теория алгоритмов, основные открытия и приложения / Успенский В.А, Семенов А.Л. - М.: Наука, 1987. – 288 с.

[Хендерсон, 1983] Хендерсон, П. Функциональное программирование. Применение и реализация : пер. с англ / Хендерсон П. – М.: Мир, 1983 – 349 с.

[Хювен и др, 1990] Хювенен, Э. Мир лиспа: Введение в язык Лисп и функциональное программирование : пер. с финск., в 2 т. / Э. Хювенен, Й. Сеппянен. – М.: Мир, 1990. – 1 т.

[NENO, 2011] Проект NENO [Электронный ресурс]. – 2011. – Режим доступа: http://neno.lanl.gov/Home.html - Дата доступа: 20.12.2010

[OSTIS, 2011] Открытая семантическая технология проектирования интеллектуальных систем [Электронный ресурс]. – 2011. - Режим доступа: http://ostis.net. – Дата доступа: 10.01.2011

[Rosenberg, 1996] How Debuggers Works: Algorithms, Data Structures, and Architecture / Jonathan B. Rosenberg. – Wiley, 1996. – 272 p.