OSTIS-2012
(Open Semantic Technologies for Intelligent Systems)

УДК 004.822:514

ОНТОЛОГИЧЕСКАЯ ПАРАДИГМА ПРОГРАММИРОВАНИЯ

Грибова В.В., Клещев А.С.

Учреждение Российской академии наук Институт автоматики и процессов управления Дальневосточного отделения РАН (ИАПУ ДВО РАН), Владивосток, Россия

gribova@iacp.dvo.ru,
kleschev@iacp.dvo.ru

Данная работа посвящена описанию новой парадигмы программирования – онтологической, которая является дальнейшим развитием парадигмы декларативного программирования. В работе описана постановка задачи, модель данных, лежащая в основе парадигмы, основные конструкции языка.
Ключевые слова: парадигмы программирования, модель данных, семантические сети, онтология.

ВВЕДЕНИЕ

Трудоемкость разработки программных систем сопоставима (а порой и превышает) сложность разработки крупных материальных объектов [Обзор Microsoft, 2005]. При этом сопровождение таких систем является еще более дорогим процессом [Промыслов и др., 2005], что требует от исследователей поиска новых подходов, направленных на решение этих проблем. В качестве одного из таких подходов была предложена парадигма декларативного программирования и разработанные в ее рамках языки декларативного программирования.

Программа на языке декларативного программирования в общем случае представляет собой описание абстрактной модели решаемой задачи или, согласно [Дехтяренко, 2003], исполнимую спецификацию результата вычислений по программе. Программисту не требуется описывать процесс управления вычислениями, он полностью возлагается на процессор языка. Среди множества преимуществ парадигмы декларативного программирования, в качестве основных называются: более простое написание программы, более легкое ее понимание по сравнению с соответствующими императивными программами, упрощение ее сопровождения и модификации [Lloyd, 1994].

К языкам декларативного программирования относят, как правило, языки функционального и логического программирования. Однако, как отмечено в [Lloyd, 1994], провозглашенная идея декларативного программирования в современных языках логического и функционального программирования в полной мере не реализована; в результате практические программы на Прологе или Лиспе ненамного легче разрабатывать, понимать и сопровождать, чем программы на императивных языках. Среди других недостатков языков декларативного программирования называют скудные средства взаимодействия с пользователем и сложность внесения в программы императивных операторов, если их использование более удобно по сравнению с декларативными конструкциями.

Цель данной работы – предложить новую парадигму программирования, названную парадигмой онтологического программирования, как дальнейшее развитие парадигмы декларативного программирования, а также предложить язык программирования OPL (Ontological Programming Language), удовлетворяющий определению декларативного программирования, в котором программа является онтологией результата вычислений; для этого языка также вводятся механизмы организации интерфейса с пользователем и включения в язык императивных конструкций.

1. Постановка задачи

При решении любой задачи на компьютере результат может быть получен лишь как итог некоторого процесса вычислений, который будем называть процессом получения результата. Сложность написания программы для решения некоторой задачи в значительной степени состоит в том, что разработчик этой программы должен представить себе все множество процессов получения результатов решения задачи (экстенсионал задачи) при различных допустимых исходных данных и определить это множество

средствами языка программирования в виде программы [Успенский и др., 1987]. Сложность понимания программы сопровождающим программистом в значительной степени состоит в том, что этот программист должен по программе представить себе экстенсионал задачи и понять, почему процессы получения результата ведут именно к результату решения этой задачи. Сложность модификации программы в значительной степени состоит в том, что сопровождающий программист, понимая, как должны измениться результаты решения задачи, должен понять, как должны измениться процессы получения результатов, чтобы они приводили именно к этим изменениям результатов, а затем понять, как эти изменения в процессах получения результатов могут быть выражены через изменения в программе. Рассмотрим с этой точки зрения, как связаны процессы получения результата и программы с результатами вычислений в парадигмах императивного, функционального и логического программирования.

В основе парадигмы императивного программирования лежат вычислительные модели [Себеста, 2001, Floyd, 1979]. В этих моделях процесс получения результата представляет собой последовательность состояний, первое из которых получается из исходных данных с помощью входной процедуры, каждое следующее состояние получается из предыдущего с помощью оператора непосредственной переработки, а из последнего состояния извлекается результат вычислений с помощью выходной процедуры. Все состояния вычислительного процесса, кроме последнего, лишь косвенно связаны с результатом вычислений (решения задачи). В современных императивных языках состоянием вычислительного процесса является множество значений переменных, а оператором непосредственной переработки – оператор присваивания, с помощью которого следующее состояние процесса получается изменением значения одной переменной, а все остальные переменные сохраняют свои значения. В программе на императивном языке последовательности выполнения операторов присваивания определяются с помощью линейных участков, условных операторов и операторов цикла (а также вызовов процедур).

В основе парадигмы функционального программирования лежит ламбда-исчисление [Себеста, 2001, Floyd, 1979]. Процесс получения результата может быть представлен ориентированной размеченной сетью вызова функций, в которой меткой каждой терминальной вершины является некоторое исходное данное, а меткой каждой нетерминальной вершины – значение некоторой функции, аргументами которой являются метки концов дуг, выходящих из этой вершины (ориентация дуг – от результата к аргументу). Результатом вычислений является метка корня сети (вершины, в которую не входит ни одна дуга). Все промежуточные значения (метки нетерминальных вершин), кроме метки корня, лишь косвенным образом связаны с результатом вычислений. В современных функциональных языках имеется некоторые наборы базовых функций, а также средства определения (конструирования) новых функций из базовых и уже определенных (в том числе условные термы и рекурсия).

В основе парадигмы логического программирования лежит исчисление предикатов первого порядка [Себеста, 2001, Floyd, 1979]. Процесс получения результата может быть представлен ориентированной размеченной сетью вывода результата, в которой меткой каждой терминальной вершины является кортеж отношения, представляющий некоторое исходное данное, а меткой каждой нетерминальной вершины – кортеж отношения, являющийся результатом применения некоторого правила к посылкам, которыми суть метки концов дуг, выходящих из этой вершины (ориентация дуг – от заключения к посылкам). Результатом вычислений (вывода) является метка корня сети. Все промежуточные значения (метки нетерминальных вершин), кроме метки корня, лишь косвенным образом связаны с результатом вычислений. В современных логических языках программа представляет собой некоторый запрос и множество правил (импликаций) и фактов (кортежей отношений).

В каждой из трех рассмотренных парадигм результат вычислений получается лишь на последнем шаге процесса получения результата, а все остальные шаги этого процесса имеют к результату вычислений лишь косвенное отношение, на них получаются лишь некоторые промежуточные значения. Такие процессы получения результата можно назвать косвенными по отношению к результату.

Таким образом, для снижения сложности разработки программ, их понимания и модификации необходимо предложить такую парадигму программирования, в которой процессы получения структурных результатов вычислений, представляемых в виде ориентированных графов, были бы прямыми, т.е. состояли из шагов, на каждом из которых формируется некоторая часть результата. В этом случае, сам такой процесс получения результата может быть представлен в виде графа, совпадающего с этим результатом, а значит, программа на языке является (исполнимой) спецификацией множества результатов вычислений (а не множества косвенных процессов их получения). Разрабатывая такую программу, программист должен лишь специфицировать множество результатов вычислений; анализируя такую программу, программист должен представить себе лишь множество результатов вычислений, специфицируемых этой программой; модифицируя такую программу, программист должен лишь понять, как следует изменить спецификацию множества результатов вычислений, чтобы получить

требуемые изменения этих результатов, причем каждое такое изменение в программе является локальным, связанным с изменением соответствующих частей результатов вычислений. Такая спецификация множества результатов вычислений, в соответствие с современными представлениями в области искусственного интеллекта, называется онтологией результата вычислений. Учитывая это, назовем предлагаемую парадигму ***парадигмой онтологического программирования***. Ее можно рассматривать в качестве более полного воплощения парадигмы декларативного программирования по сравнению с парадигмами функционального и логического программирования.

2. Модель данных, лежащая в основе парадигмы

Одно из положений парадигмы онтологического программирования состоит в том, что все данные имеют вид семантических сетей. Для определенности рассматриваются иерархические семантические сети (что не является принципиальным), определяемые следующим образом. Иерархическая семантическая сеть есть связный ориентированный граф без циклов, в котором каждая дуга имеет метку, а вершины могут быть одного из двух типов - простые и структурные; вершина сети, в которую не входит ни одна дуга, называется ее корнем. Каждая простая терминальная вершина сети имеет в качестве метки константу некоторого сорта. Корень сети имеет две метки-термина, из которых первая метка есть метка класса (имя функции, в результате выполнения которой была создана эта семантическая сеть), а вторая - индивидуальная метка этой сети. Каждая структурная вершина сети является контейнером, содержащим упорядоченное конечное множество иерархических семантических сетей с одной и той же меткой класса и попарно различными индивидуальными метками.

Семантическая сеть может быть постоянного хранения (хранимая независимо от приложения) для обработки всеми приложениями, реализованными в рамках парадигмы онтологического программирования, так и временная, создаваемая при запуске приложения и прекращающая свое существование по окончании этого запуска.

Использование семантических сетей в качестве модели данных означает, что обрабатываются только целостные информационные структуры, которые выделяются к качестве объектов обработки, в отличие от других парадигм, поддерживающих обработку информационных объектов, хотя и произвольной структуры, но не обязательно объединенных в целостную информационную структуру (как правило, целостные информационные структуры разделяются на различные фрагменты, являющиеся значениями различных переменных, связь между которыми известна и понятна только разработчику программы). Например, в экспертной системе медицинской диагностики такими целостными информационными структурами являются история болезни, база знаний и объяснение, представленные в форме семантических сетей.

3. Приложение

В общем случае приложение имеет одну или несколько выходных семантических сетей (результатов), может иметь (или не иметь) одну или несколько входных семантических сетей (входных данных), а также иметь (или не иметь) промежуточные семантические сети (промежуточные результаты). Каждая промежуточная или выходная семантическая сеть приложения вычисляется с помощью функции (название которой является меткой класса корня этой семантической сети).

Каждая функция может иметь (или не иметь) входные семантические сети, в точности одну выходную семантическую сеть и не может иметь промежуточных семантических сетей

Входные семантические сети, как приложения, так и функций, могут быть постоянного хранения (созданными и существующими независимо от приложения), так и временными созданными во время запуска приложения (как результат одной из его функций). Соответственно, выходные семантические сети, как приложения, так и функций, могут быть как постоянного хранения, так временными.

Между всеми семантическими сетями приложения (входными, промежуточными и выходными) задается частичный порядок, в котором вычисляются промежуточные или выходные семантические сети, как результаты соответствующих функций. Поэтому программу приложения можно рассматривать как суперпозицию этих функций, связанных с вычислением всех семантических сетей приложения, не являющихся входными семантическими сетями постоянного хранения.

Таким образом, выполнение приложения состоит в вычислении выходных семантических сетей по входным семантическим сетям постоянного хранения, созданным до запуска приложения (их может и не быть) через промежуточные семантические сети, создаваемые при запуске приложения.

Каждая функция приложения представляет собой онтологию (описание структуры) выходной семантической сети этой функции на языке OPL.

4. Язык онтологического программирования

Язык OPL является визуальным логическим языком программирования. Описание функции – онтология выходной семантической сети,

представляет собой ориентированный граф с размеченными вершинами и дугами. Метками дуг являются термины, которые в ходе выполнения программы становятся метками дуг выходной семантической сети функции. Метками вершин являются логические формулы.

Процесс вычисления выходной семантической сети функции сводится к построению этой сети, начиная с корня. В процессе построения семантической сети устанавливается однозначное соответствие между вершинами и дугами этой сети и вершинами и дугами ее онтологии (описанию функции). Выходная семантическая сеть, в зависимости от описания функции, может формироваться как автоматически, без участия пользователя, так и интерактивно, с участием пользователя, который в процессе диалога управляет процессом построения этой сети.

Логическими формулами языка являются: простая формула, простая кванторная формула, унарная формула, пропозициональная формула, структурная кванторная формула, множество импликаций.

Простая формула есть граф, состоящий из единственной вершины с меткой c. Метка c является константой одного из сортов, либо переменной v, либо значением переменной v*.

Унарная формула есть граф, состоящий из начальной вершины и дуги с меткой T (термином), выходящей из начальной вершины и входящей в начальную вершину некоторой логической формулы F.

Пропозициональная формула есть граф, состоящий из начальной вершины с пропозициональной меткой P и выходящих из нее n дуг (не менее двух), каждая из которых имеет метку Ti (термин; i от 1 до n и все эти метки должны быть попарно различны) и входит в начальную вершину некоторой логической формулы Fi. Пропозициональными метками P являются: & — конъюнкция, ∨ — дизъюнкция, | — исключающее или. Множество пропозициональных меток должно быть расширяемым.

Простая кванторная формула есть граф, состоящий из единственной вершины с меткой QMT, где Q - знак квантора, M - описание множества, а T – термин. Знаком квантора может быть: ∀ (для всех), ∃ (существует), ∃2 (существует не менее двух), ∃? (существует, но не для всех), ∃! (существует и единственен), ∃[] (существует подъинтервал).

Описание множества может явно задаваться либо значениями своих элементов, которыми являются попарно различные константы, либо определяться сортом возможных значений элементов множества, либо целым или вещественным интервалом. В качестве описания множества также может выступать переменная. Названием сорта может быть: "строка", "целый", "вещественный", "целый интервал", "вещественный интервал", "дата-время". Множество сортов и кванторов является расширяемым.

В зависимости от описания множества возможны различные кванторы, предшествующие его описанию. Описание множества и кванторы задают условия и ограничения при порождении очередного фрагмента результата.

Структурная кванторная формула состоит из начальной вершины с меткой, имеющей вид QMT, где Q — знак квантора, M — описание множества, а T — термин, и логической формулы F, начальная вершина которой изображается внутри начальной вершины структурной кванторной формулы

Множество импликаций есть граф, состоящий из одной вершины с меткой {П1 ⇒ F1, …, Пm ⇒ Fm}, где П1, …, Пm –антецеденты, а F1, …, Fm – консеквенты импликаций. Антецедент импликации есть конечное множество компонент - логических формул, каждая из которых может иметь префикс. Префикс есть название некоторой программы на языке. Консеквент импликации есть логическая формула.

Переменные, которые объявлены в логических формулах, могут входить только в антецеденты и консеквенты импликаций, а специальные метки вершин в унарных формулах — только в антецеденты импликаций. Если переменная входит в консеквент импликации, то она должна входить и в ее антецедент.

Переменные используются в импликации, если очередной фрагмент результата должен быть сформирован в зависимости от значений других меток (терминов) уже сформированных фрагментов графа, либо внешних по отношению к данному (таким образом могут использоваться внешние данные и эти данные также представлены в форме графа.). В этом случае в качестве метки логической формулы либо описания множества (зависит от типа логической формулы) выступает переменная. При этом если при формировании графа используются внешние данные по отношению к данному графу, то эта формула содержит префикс, совпадающий с именем файла, из которого будут вычисляться значения переменных.

В общем случае при порождении результата вычислений, когда активной вершине является множество импликаций, то вычисляется истинность антецедентов этих импликаций. Если у некоторой импликации истинен ее антецедент, то активной вершине соответствует начальная вершина консеквента этой импликации.

Если в антецеденте импликации в логических формулах присутствует переменная, то при порождении вместо имени переменной осуществляется подстановка ее значений. Импликация будет выполняться если при подстановке некоторого значения переменной все компоненты антецедента истинны.

Для организации сложных эффективных вычислений средств логического языка может оказаться недостаточно. Для решения этой проблемы язык OPL дополнен вычисляемыми предикатами. Вычисляемые предикаты предназначены для описания вычислительных алгоритмов, описание которых средствами декларативного логического языка слишком громоздко. Сами вычисляемые предикаты описываются средствами императивного языка в соответствующей среде создания приложений. Вычисляемый предикат имеет имя и множество формальных параметров. Вызов вычисляемого предиката с фактическими параметрами может быть лишь компонентой антецедента импликации. При поиске по образцу, когда в антецеденте импликации встречается вызов вычисляемого предиката, происходит его выполнение: в тело предиката подставляются значения, указанные в его аргументах, и производятся вычисления.

Команды абстрактного интерфейса определяют функциональность пользовательского интерфейса. Их можно разделить на четыре класса:

- команда вывода (процессором языка) сформированного фрагмента семантической сети (выполняются при построении семантической сети в интерактивном режиме или по запросу пользователя);

- команды непосредственного ввода с клавиатуры одного значения заданного типа и изменение этого значения (входит в операционную семантику кванторных формул с квантором существования и единственности, если множество, по которому пробегает квантор, бесконечно);

- команды непосредственного ввода с клавиатуры нескольких значений заданного типа и изменения этих значений (входит в операционную семантику кванторных формул с различными вариантами квантора существования, если множество, по которому пробегает квантор, бесконечно);

- команды выбора значения либо подмножества значений из предлагаемого множеств и изменение этого выбора (входит в операционную семантику дизъюнкции и кванторных формул с различными вариантами квантора существования, если множество, по которому пробегает квантор, конечно).

Команды абстрактного интерфейса ввода и выбора значений совпадают для кванторных формул с различными вариантами квантора существования, если квантор пробегает конечное множество, и пропозициональных формул дизъюнкции и исключающего или.

С использованием языка описана экспертная система медицинской диагностики. Входными данными для нее являются «Простая база наблюдений» и «История болезни». Само приложение (программа на языке) представляет собой семантическую сеть «Объяснение» (формирование объяснения). Объяснение состоит из двух частей – объяснения гипотезы о том, что пациент здоров, и объяснения гипотез о том, что пациент болен одним из заболеваний, представленных в базе заболеваний (см. рис. 1).

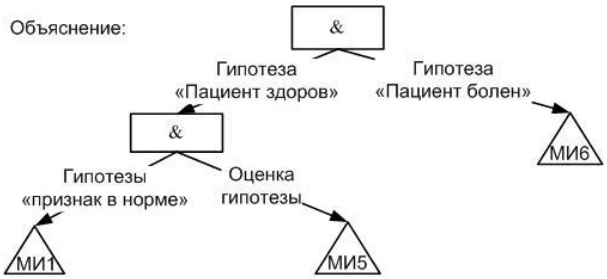

Рисунок 1 – Объяснение экспертной системы медицинской диагностики

Объяснение гипотезы о том, что пациент здоров, состоит из объяснения гипотез о том, что все наблюдаемые признаки пациента находятся в норме, и из оценки гипотезы о том, что пациент здоров.

Импликация МИ1 (рис.2) формирует значение переменной v1, которым является множество всех наблюдаемых признаков в истории болезни; для каждого из этих признаков объяснение гипотезы о том, что он находится в норме, состоит из объяснения гипотез о том, что все наблюдавшиеся значения этого признака являются нормальными, и из оценки гипотезы о том, что этот признак находится в норме.

Рисунок 2 – Описание гипотезы "признак в норме"

Импликация МИ2 (рис. 3)для каждого значения v1* переменной v1 формирует значение переменной v2, которым является множество моментов наблюдения признака v1* в истории болезни; для каждого из этих моментов наблюдения формируется оценка гипотезы о том, что значение признака v1*, наблюдавшееся в этот момент, является нормальным.

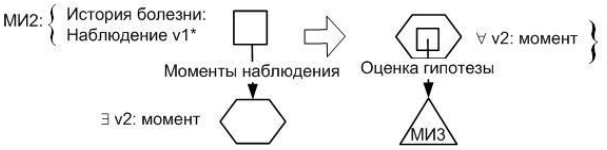

Рисунок 3 – Описание гипотезы "значение нормальное"

Первая импликация (рис. 4) из множества МИ3 для каждого значения v1* переменной v1 и для каждого значения v2* переменной v2 формирует значение переменной v3, которым является значение, наблюдавшееся в момент v2* у признака v1* в истории болезни, а также формирует значение переменной v4, которым является множество

нормальных значений признака v1* в базе заболеваний, а затем выполняется вычисляемый предикат «принадлежность» (принадлежность элемента v3 множеству v4); если значение этого предиката есть «истина», то формируется оценка «подтверждена» гипотезы о том, что значение признака v1*, наблюдавшееся в момент v2*, является нормальным. Вторая импликация, если значение этого предиката есть «ложь», формирует оценку «опровергнута» гипотезы о том, что значение признака v1*, наблюдавшееся в момент v2*, является нормальным.

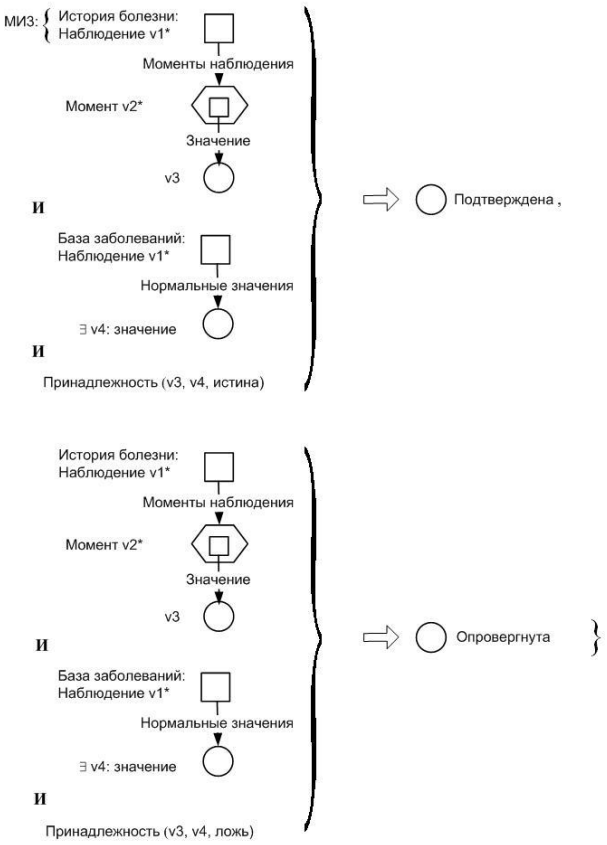

Рисунок 4 – Описание оценки гипотезы "значение нормальное"

Первая импликация (Рис. 5) из множества МИ4 для значения v1* переменной v1 формирует оценку «подтверждена» гипотезы о том, что признак v1* находится в норме, если в объяснении оценки гипотез о том, что значения признака v1*, наблюдавшееся в моменты, равные всем значениям переменной v2, являются нормальными, «подтверждена». Вторая импликация, для значения v1* переменной v1 формирует оценку «опровергнута» гипотезы о том, что признак v1* находится в норме, если в объяснении оценки гипотез о том, что значения признака v1*, наблюдавшееся в моменты, равные некоторым элементам множества - значения переменной v2, являются нормальными, «опровергнута».

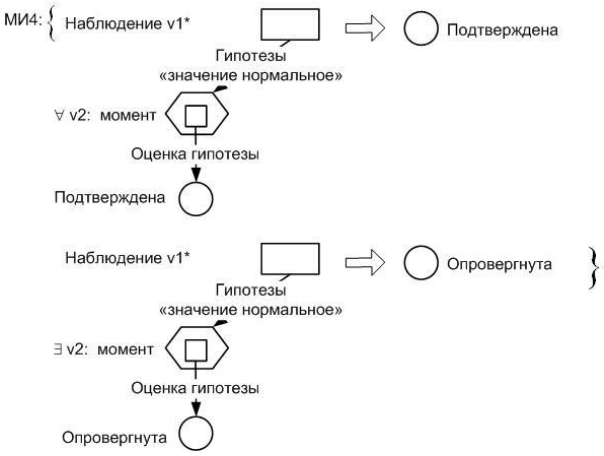

Рисунок 5 – Описание оценки гипотезы "признак в норме"

Первая импликация из множества МИ5 (рис. 6) формирует оценку «подтверждена» гипотезы о том, что пациент здоров, если в объяснении оценка гипотезы о том, что признаки, равные всем значениям переменной v1, есть «подтверждена». Вторая импликация формирует оценку «опровергнута» гипотезы о том, что пациент здоров, если в объяснении оценка гипотезы о том, что признаки, равные некоторым элементам множества - значения переменной v1, есть «опровергнута».

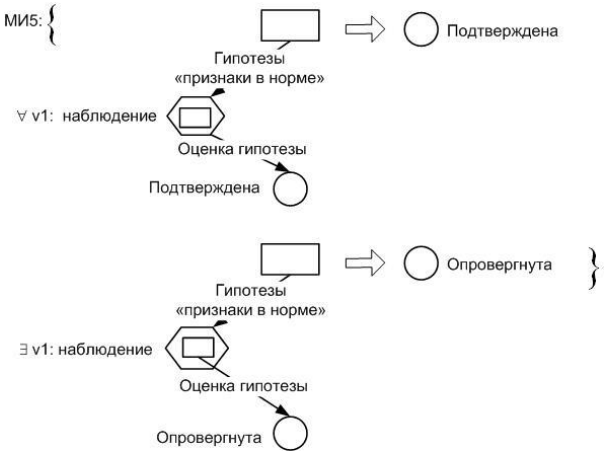

Рисунок 6 – Оценка гипотезы "пациент здоров"

Гипотеза «пациент болен» описывается аналогичным образом.

5. Приложение

Основные задачи, которые были поставлены при разработке новой онтологической парадигмы программирования – это снижение сложности разработки и сопровождения программ. Анализ существующих парадигм и языков программирования показал, что основными сложностями, с которыми сталкиваются программисты, разрабатывающие программы и программисты, которые их сопровождают, являются:

- сложность программисту, разрабатывающему программу, понять, формализовать и описать

средствами языка программирования множество процессов получения результатов решения задачи при различных исходных данных; эта сложность имеет место, независимо от того, что представляет собой этот процесс – последовательность состояний, сеть вызова функций либо сеть вывода результата;

- сложность программисту, сопровождающему программу, понять, как устроено описанное программистом множество решений, а в случае, если его необходимо изменить, понять, как внести все необходимые изменения в программу;

- сложность программисту реализовать пользовательский интерфейс, обеспечивающий ввод информации от пользователя и вывод результатов и встроить его в процесс получения результата; в результате программа становится еще более сложной для понимания и сопровождения, а изменение процесса вычислений приводит к необходимости внесения изменения в интерфейс и наоборот.

Парадигма онтологического программирования предлагает следующие решения указанных выше проблем.

1. Использование семантических сетей в качестве модели данных означает, что обрабатываются только целостные информационные структуры. Это упрощает сопровождение приложений – программисту легче понять целостную информационную структуру, нежели множество разрозненных структур, связь между которыми известна и понятна только разработчику программы.

2. Приложение в рамках предложенной парадигмы может быть представлено как частично упорядоченное множество вызовов функций; каждая функция формирует в качестве результата и, соответственно, обрабатывает в качестве входных данных только целостные информационные структуры, которых, как правило, с приложением связано не слишком много. Соответственно, основная «нагрузка» ложится на описание каждой функции. Описание функции представляет собой онтологию результата функции. Таким образом, при разработке приложения необходимо: выделить целостные информационные структуры, определить частичный порядок между задачами их формирования; описать функции для вычисления каждой такой информационной структуры. Сопровождение приложения сводится к добавлению новых целостных информационных структур и функций, изменению частичного порядка вызовов функций приложения и сопровождению самих функций - онтологий их результатов. Описание онтологии результата, как правило, более компактно и наглядно, чем описание процесса его получения.

3. Онтология ансамбля семантических сетей представляется на визуальном языке в виде ориентированного графа. Графовое (визуальное) представление согласно многим исследованиям, например [Касьянов и др., 2003], также упрощает понимание программы. Таким образом, программа на языке OPL, в общем случае, является более понятной, чем программы, представленные на языках других парадигм.

4. Модель данных, поддерживаемая парадигмой, включает данные (семантические сети) постоянного хранения (подобно базам данных). Это, во-первых, упрощает их повторное использование; во-вторых, упрощает разработку программ, поскольку не требуется дополнительного программирования для чтения информации и ее записи в поддерживаемый формат хранения.

5. Семантические сети образуют ансамбль, для которого программа функции является онтологией. Тем самым, многие функции оказываются повторно-используемыми. При этом если процесс формирования информационной структуры является интерактивным, то каждая функция может рассматриваться как редактор формирования такой структуры. Время разработки приложения за счет повторного использования функций может быть сокращено. Например, если функция представляет собой редактор историй болезни, то она может быть использована для ввода историй болезни в экспертных системах медицинской диагностики, в медицинских диагностических тренажерах, во многих других медицинских информационных системах. При этом для каждой такой системы не надо разрабатывать редактор для формирования историй болезни, что, в общем случае, является трудоемкой работой.

6. Абстрактный пользовательский интерфейс входит в операционную семантику логических формул языка. Таким образом, не требуется программирование интерфейса функций.

7. Процесс получения результата в предлагаемой парадигме до некоторой степени аналогичен такому же процессу в логической парадигме. Однако в логической парадигме на каждом шаге процесса вывода должно определяться множество применимых правил и множество значений их посылок. В онтологической парадигме процесс порождения является детерминированным, для каждого шага процесса построения семантической сети определена единственная формула, операционная семантика которой определяет действия на этом шаге.

В настоящее время сложно обозначить весь спектр применений парадигмы онтологического программирования. На сегодня видны два основных применения онтологической парадигмы программирования - создание интеллектуальных систем и редакторов информационных ресурсов любого типа – онтологий, знаний, данных.

Работа выполнена при финансовой поддержке РФФИ, проект 10-07-00090

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

[Обзор Microsoft, 2005] Обзор Microsoft // 2005. [Электронный ресурс] http://www.ict.edu.ru/ft/005126/intro_net.pdf.

[Промыслов и др., 2005] Промыслов В.Г., Жарко Е.Ф., Промыслова О.А. Практические аспекты сопровождения и модификации сложных программных систем // Труды IV Международной конференции "Идентификация систем и задачи управления" SICPRO'05. Москва, 25-28 января 2005 г. М.: ИПУ РАН, 2005. С. 1151.

[Дехтяренко, 2003] Дехтяренко И.А. Программирование в повествовательном наклонении // SoftCraft. Декларативное программирование. 04.02.2003. [Электронный ресурс] - http://www.softcraft.ru/paradigm/dp/dp01.shtml

[Lloyd, 1994] Lloyd J.W. Practical Advantages of Declarative Programming // Proceedings of the 1994 Joint Conference on Declarative Programming.

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

[Себеста, 2001] Себеста Роберт В. Основные концепции языков программирования = Concepts of Programming Languages. - 5-е изд. - М.: «Вильямс», 2001. — С. 672.

[Floyd, 1979] Floyd R.W. The Paradigms of Programming // Communications of the ACM, 1979. 22(8). Pp. 455-460.

[Касьянов и др., 2003] Касьянов В.Н., Евстигнеев В.А. Графы в программировании: обработка, визуализация и применение. – Изд-во: БХВ-Петербург, 2003. - 1104 с.

ONTOLOGICAL PROGRAMMING PARADIGM

Gribova V., Kleschev A.

Institute of Automation and Control Processes, Far Eastern Branch of RAS

gribova@iacp.dvo.ru, kleschev@iacp.dvo.ru

This report is devoted to a new ontological programming paradigm. It is an evolution of the declarative programming paradigm. The data model, basic structures of language, and examples are described.

Key words: programming paradigm, data model, semantic network, ontology

INTRODUCTION

Software development is time consuming process. Even more difficult is software maintenance. They both requires new approaches from developers to resolve these problems. The declarative programming paradigm and languages realized this paradigm were proposed as an approach to solve the problem mentioned above.

In general a program in a declarative language is a description of an abstract model of the task to be solved, or, in accordance with [3], an executing specification of a result of computing. The programmer does not have to describe a process to control computation, it is the function of the language processor. Among other advantages is the fact that it is easer to write, to understand, and to maintain programs using languages of the paradigm in comparison with those written in imperative languages.

Declarative languages comprise logical and functional languages. However, the main idea of declarative programming in the modern logical and functional languages has not been realized yet completely. As a result programs written by Prolog and Lisp are not considerably easier to develop, understand, and maintein than programs written in imperative languages. Among other drawbacks of declarative languages are poor facilities for user interface realization, difficulty of including imperative operators, if necessary.

The aim of this report is to suggest a new programming paradigm, called the ontological programming paradigm as a further evolution of the declarative programming paradigm and to suggest an ontological programming language OPL, satisfying the declarative programming definition where a program is an ontology of results of computing. Mechanisms for user interface realization and for including imperative structures are proposed.

MAIN PART

According to the suggested paradigm the process of obtaining results is a graph, a program in the OPL is an executive specification of the set of results of computation. This specification is an ontology of results.

All data in the ontological paradigm are semantic networks. The semantic network may be stored constantly (out of the programs) or temporarily (within the program). Using semantic networks as a data model means that objects of processing are complete information structures.

The OPL is a visual logical language of programming. Logical formulas of the language are: a simple formula, a simple quantifier formula, a unary formula, a propositional formula, a structural quantifier formula, a set of implications.

For realization of complicated calculations computed predicates are added in the OPL. Their purpose is to describe calculations using operators of an imperative language (for example, Java). A computed predicate has name and a set of formal parameters. Call of a computed predicate with a set of actual parameters can be only from an antecedent of an implication.

Abstract interface commands define functional of a user interface. They are divided into four classes: output commands, input (edit) commands of a value of a type, input (edit) commands of a set of values of a type, choice (edit) a subset from the set of values.

CONCLUSION

The new paradigm is intended for reducing labour-intensiveness of development and maintenance of intelligent systems. The main idea is to describe an ontology of results using the visual logical language of programming. The programmer does not have to describe a process to control computation, it is the function of the language processor. All data in the ontological paradigm are semantic networks. The language has facilities for user interface realization and for including imperative structures.