УДК 681.3.06

ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ ДЛЯ РЕШЕНИЯ ТРУДНОФОРМАЛИЗУЕМЫХ ЗАДАЧ

В.С. Выхованец
Институт проблем управления им. В.А. Трапезникова РАН
Россия, 117997, Москва, Профсоюзная ул., 65
E-mail: valery@vykhovanets.ru

М.И. Говоров
МГТУ им. Н.Э. Баумана
Россия, 105005, Москва, 2-я Бауманская ул., 5, стр. 1
E-mail: makcimgovorov@gmail.com

Ключевые слова: методологии и технологии программирования, семантический разрыв, трудноформализуемые задачи, языково-ориентированное программирование, предметно-ориентированные языки, языковые инструментальные средства, контекстная технология программирования, представление знаний

Аннотация: выделен класс задач, названных трудноформализуемыми, для решения которых целесообразно создание индивидуальных языков программирования, рассмотрены известные технологии языково-ориентированного программирования, определены их достоинства и недостатки при решении трудноформализуемых задач.

1. Введение

За последние три десятка лет в программной инженерии можно увидеть появление и развитие множества методологий и технологий разработки программного обеспечения [1]. За столь короткий период языки программирования трансформировались от примитивных ассемблеров до объектно-ориентированных языков. Прослеживается основная тенденция, связанная с увеличением уровня абстракции языков программирования, вызванная постоянным увеличением сложности решаемых задач и соответствующим этому увеличению объемам используемых вычислительных ресурсов.

Повышение уровня абстракций языков программирования снимает только часть проблем, не затрагивая существенным образом понятийный аппарат самого языка программирования [2]. По этой причине выразительные возможности широко используемых языков программирования так и остались привязанными к встроенной в эти языки примитивной системе операций над простыми типами данных.

В итоге, проектирование информационных систем до сих пор выполняется в основном на интуитивном уровне с применением неформализованных методов, основанных на искусстве разработчиков, их практическом опыте, экспертных оценках и дорогостоящих экспериментальных проверках получаемых результатов.

Движущая сила наблюдаемого бурного развития методологий и технологий программирования – стремление сократить семантический разрыв между средствами первичного описания предметной области и языками программирования [3]. Последствиями семантического разрыва являются высокая стоимость разработки программных

средств, их низкая эффективность и надежность, объективная трудоемкость, интеллектуальная и технологическая сложность самого процесса программирования.

Настоящий доклад посвящен обзору известных методов и средств создания предметных языков программирования, наиболее эффективным образом сокращающих семантический разрыв между языковыми средствами постановки задач и языковыми средствами, используемыми для их решения.

2. Трудноформализуемые задачи

Все чаще перед разработчиками программ возникают задачи, которые можно назвать трудноформализуемыми [4]. Основной особенностью трудноформализуемых задач является наличие их общей формальной постановки (эффективно распознаваемых условий), которая, тем не менее, не имеет единого алгоритма решения. В этом случае для каждого конкретного варианта трудноформализуемой задачи (для каждого набора входных данных) приходится искать свой, присущий только этому варианту (этому набору данных) алгоритм решения.

Примерами трудноформализуемых задач могут служить задачи аннотирования текстов, доказательства теорем, распознавания образов, принятия решений, классификации и кластеризации данных и т.п. Классическим примером трудноформализуемой задачи является задача тестирования программных средств. Для ее решения приходится создавать специализированную программу тестирования, привязанную тестируемому программному средству. Следует заметить, что существует множество подходов к тестированию программ, но эффективное тестирование сложных программных средств – это процесс в высшей степени творческий, не сводящийся к следованию строгим методикам.

Трудноформализуемые задачи могут быть охарактеризованы большими трудозатратами, которые необходимы для их решения. Такие задачи, как правило, требуют использования большого объема привлекаемых знаний об используемой предметной области и учета индивидуальных особенностей решаемой задачи.

Решение трудноформализуемых задач основано на изучении предметной области под углом зрения некоторой конкретной прикладной проблемы, конкретизирующей трудноформализуемую задачу до ее разрешимого частного случая. Именно в процессе такого изучения и появляется понимание специфики решаемой задачи, а выражение этой специфики на языке программирования позволяет получить искомое решение.

Поиск решений трудноформализуемых задач сопровождается выдвижением множества гипотез, их теоретическим и экспериментальным обоснованием, поиском и проверкой эвристических правил, не имеющих непосредственного теоретического обоснования. Возможна также ситуация, при которой для каждого набора входных данных, необходимо искать и обосновывать свой алгоритм решения задачи.

Общим недостатком известных технологий программирования является практическое игнорирование того факта, что трудноформализуемую задачу не опишешь понятиями, встроенными в универсальные языки программирования.

Последнее приводит к появлению труднопреодолимого семантического разрыва, который возникает между содержательными представлениями разработчика, выраженными в терминах предметной области, и ее формальной моделью на языке программирования. Проявление семантического разрыва, например, в области программной инженерии, заключается в том, что результатом изучения программы является знание отдельного индивида, которое не отчуждается от индивида и легко им теряется [5].

3. Предметно-ориентированные языки

Самым естественным методом сокращения семантического разрыва видится приближение выразительных средств языков программирования к языкам, близким к языкам исходного описания предметной области. В этом случае понятия языка программирования должны совпадают или быть очень близкими понятиям, примененным для описания решаемой прикладной задачи. А это закономерно приведет к повышению качества, производительности и эффективности разработки программных средств.

Одной из самых распространенных практик такого приближения является использование языково-ориентированного подхода, при котором предполагается создание для каждой предметной области своего предметно-ориентированного языка.

Предметно-ориентированный язык (англ. Domain Specific Language, DSL) – это язык программирования с ограниченными выразительными возможностями, ориентированный на решение задач в некоторой конкретной предметной области [6]. При использовании языково-ориентированного подхода решение задачи разбивается на два этапа: создание специализированного языка, приближенного к предметной области, и создание программного средства на этом языке.

Сильная связь предметно-ориентированных языков с предметной областью позволяет разработчику оперировать понятиями, близкими к этой предметной области, выражать решения прикладных задач в более естественном виде, тем самым заметно упрощая написание сложных программ.

В идеале при использовании предметно-ориентированных языков решение прикладной задачи может быть описано специалистами предметной области, не обладающим знаниями языков программирования общего назначения. Это позволяет заметно сократить возникающий при разработке программ семантический разрыв между предметной областью и языком программирования.

Примерами предметно-ориентированных языков могут служить такие языки как: SQL (англ. Structured Query Language) – язык, применяемый для создания, модификации и управления данными в реляционных базах данных [7]; язык регулярных выражений [8]; XPath – язык запросов к элементам XML-документа [9]; языки командных процессоров операционных систем [10]; различные математические языки (MathCAD, Matlab, Wolfram Mathematica); языки систем автоматизированного проектирования, и другие.

Предметно-ориентированные языки делятся на две категории: внешние и внутренние. Внешними называются те предметно-ориентированные языки, которые написаны на языке, отличном от основного языка разрабатываемой программы. Внутренние предметно-ориентированные языки, в свою очередь, являются расширение языков общего назначения.

Основным достоинством внешних языков является полная свобода действия его разработчиков. При создании внешнего предметно-ориентированного языка разработчик начинает с «чистого листа», ему не навязан какой-либо язык, от синтаксиса и семантики которого необходимо отталкиваться: разработчик волен задать какой угодно синтаксис и семантику для создаваемого языка.

Это является как сильной, так и слабой стороной внешних предметно-ориентированных языков. Основным недостатком таких языков является необходимость привлекать много ресурсов для создания и поддержки лексических и синтаксических анализаторов, кодогенераторов (компиляторов) или интерпретаторов внешних предметно-ориентированных языков. Также возникает необходимость в создании и интегрированных сред разработки.

Если для языков общего назначения стиль и технология программирования уже давно выработаны, то для вновь создаваемых языков возникает необходимость в их разработке. Также возникает проблема связи между разными предметно-ориентированными языками, используемыми в одном проекте. По этим причинам внешние предметно-ориентированные языки целесообразно использовать только в крупных проектах [11].

Внутренние предметно-ориентированные языки являются надстройками над языками программирования общего назначения, расширяя их синтаксис и семантику под свои нужды. В этом случае разработчик внутреннего предметно-ориентированного языка может использовать возможности базового языка и все те преимущества, которые дает его интегрированная среда разработки. Однако разработка внутренних предметно-ориентированных языков ограничена синтаксисом базового языка.

Примером языка программирования, позволяющим создавать внутренние предметно-ориентированные языки, может служить гибридный язык программирования Nemerle [12]. Механизм макросов позволяет описывать на Nemerle новые языковые конструкции и использовать их наравне со встроенными. Следует заметить, что большинство управляющих конструкций этого языка (операторы if, when, циклы всех видов) реализованы в виде макросов стандартной библиотеки Nemerle.

Таким образом, внешние и внутренние предметно-ориентированные языки имеют противоположные достоинства и недостатки. Внутренние языки сильно ограничивают возможности разработчика, но создаются в рамках уже готовой технологии программирования. Внешние языки никак не ограничивают возможности разработчика, но для них требуется создавать полный комплект инструментальных средств.

На данный момент технологии языково-ориентированного программирования пока не нашли широкого применения и популярны только среди своих сторонников.

4. Языковые инструментальные средства

Проблемы в области языково-ориентированного программирования привели к созданию языковых инструментальных средств (языковых инструментариев) [13]. Под языковым инструментарием (англ. Language Workbench) понимается набор программ, который предоставляет разработчику некоторый набор возможностей по созданию и использованию внутренних и внешних предметно-ориентированных языков. Основная задача языкового инструментария – упростить создание внешних предметно-ориентированных языков программирования и интегрированных сред разработки для их использования [14].

На сегодняшний день существует большое число языковых инструментариев, сильно отличающихся своими возможностями, но в тоже время, имеющих общие черты [6]. Языковые инструментарии реализуют метаязыковый подход, при котором для описания внешнего предметно-ориентированного языка предварительно создается внутренний предметно-ориентированный язык, называемый предметным метаязыком или просто метаязыком.

По этой причине в языковых инструментариях происходит разделение процесса создания внешнего языка на два этапа. На первом этапе с помощью специализированного языка структур осуществляется определение семантической модели (синтаксического дерева) создаваемого внешнего языка: описываются типы данных и связи между ними. При этом выделяются два вида отношений между типами: агрегация и ассоциация. Отношения агрегации определяет базовую древовидную структуру модели, а отношения ассоциации задают вспомогательные горизонтальные связи. Следует заме-

тить, что создаваемая при этом модель является некоторой разновидностью диаграммы классов в нотации UML [15].

На втором этапе с помощью специализированного языка преобразований описывается поведение семантической модели (синтаксического дерева): задаются правила интерпретации синтаксических конструкций внешнего языка на некотором внутреннем (целевом) языке. Целевой язык определяется для каждого внешнего языка и должен иметь прямое отображение своих синтаксических конструкций в программы на некотором встроенном языке программирования.

В качестве примера языковых инструментариев можно привести Meta Programming System компании JetBrains [18], Software Factories компании Microsoft [19], Intentional Workbench компании Intentional Software [20]. В самом общем виде языковые инструментарии выглядят как технологически развитые генераторы компиляторов, например, такие как YACC [16] и Bison [17].

При использовании языковых инструментариев создание внешнего предметно-ориентированного языка заметно упрощается. Однако языковые инструментарии имеют ряд недостатков, которые сдерживают их применение: использование специализированных средств разработки порождает зависимость проекта от среды разработки, программа оказывается привязанной к конкретному инструменту и ее исходный текст нельзя экспортировать в другой языковой инструментарий, реализуется режим ограниченной поддержки встроенных языков программирования общего назначения.

В конечном итоге современные языковые инструментарии в большей степени нацелены не на привлечение экспертов предметной области к процессу программирования, а на повышение производительности труда самих программистов.

5. Контекстная технология программирования

В отличие от языковых инструментариев в контекстной технологии программирования использует не метаязыковый, а проязыковый подход, при котором решение каждой прикладной задачи осуществляется на индивидуальном проблемно-ориентированном (проблемном) языке, создаваемом специально для решения этой задачи [3].

Протоязык – минимальный язык, достаточный для описания онтологии (понятийной структуры), синтаксиса и семантики всех потенциально возможных проблемных языков. Основная задача протоязыка – описание понятийной структуры проблемной области, которая выделена текущей решаемой задачей, а также определение синтаксиса языковых конструкций, которые вводятся для описания способов выражения понятий в тексте. Семантическая роль протоязыка ограничена реализацией аксиомы – примитивной языковой конструкции, позволяющей ввести в использование первичные семантические категории, через которые осуществляется привязка проблемного языка к одному или нескольким внешним интерпретаторам.

Для описания понятийной структуры выделяются предметные понятия, необходимые для формулировки и решения стоящей трудноформализуемой задачи. При этом между выделенными понятиями устанавливаются два вида связей, выражающие абстракции обобщения (типизации) и ассоциации (агрегации) [21].

При обобщении происходит порождение нового понятия на основе одного или нескольких подобных понятий, когда новое понятие сохраняет общие признаки исходных понятий, но игнорирует их более тонкие различия. Частным случаем обобщения понятий является их типизация.

При ассоциации устанавливается взаимосвязь между сущностями одного и того же или разных понятий. Ассоциация выражает специфическое соединение сущностей. Это соединение позволяет от сущности одного понятия перейти к одной или нескольким сущностям других понятий. Частным случаем ассоциации понятий является их агрегация.

После описания понятийной структуры наступает этап определения языковых конструкций создаваемого проблемного языка [23]. Каждая языковая конструкция выражает некоторое понятие и состоит из последовательности понятий и термов. Если в языковой конструкции встречается некоторое понятие, то для его распознавания в тексте применяется одна из языковых конструкций этого понятия. Если в языковой конструкции встречается терм, то соответствующий ему текст должен быть непосредственно распознан во входном тексте.

Заключительным этапом является описание семантики языковых конструкций [22]. Для этого могут быть использованы только языковые конструкции, определенные ранее. Таким образом, проблемный язык описывается не поэтапно, а итерационно, причем в такой последовательности, которая позволяет новые формы выражения понятий описывать через ранее определенные.

Основная роль понятийной структуры – определить возможность выражать понятия проблемной области языковыми конструкциями других понятий. Так любая языковая конструкция, заданная для одного или нескольких обобщенных понятий, годится для выражения и понятия-обобщения. А из любого понятия-ассоциации могут быть выделены ассоциируемые в нем понятия, для распознавания которых применены выражающие их языковые конструкции.

Пример решения простейшей задачи средствами контекстной технологии приведен на листинге 1. В строке 1 вводится понятие Уравнение, с которым ассоциируются три понятия Число с именами A, B, C (ассоциируемые понятия задаются в круглых скобках после имени определяемого понятия, а обобщаемые – перед именем). В строке 2 задается синтаксис языковой конструкции для квадратного уравнения, где перечислены три понятия Число, их алиасы (ссылочные имена) в квадратных скобках, а также терминальные строки в апострофах. В строке 3 в фигурных скобках описана семантика этой языковой конструкции – извлечение из входного текста трех чисел и присваивание их ассоциируемым переменным.

В строке 4 объявляется пустое понятие (понятие без имени), для выражения которого задана одна языковая конструкция (строки 6-10). Эта конструкция применяется для решения квадратного уравнения, выраженного некоторой сущностью понятия Уравнение (строка 11). Семантика этой конструкции – нахождение решения квадратного уравнения, приведена в фигурных скобках и выражена с помощью языковых конструкций, отсутствующих этом листинге.

Листинг 1 – Решение квадратного уравнения

```
1   () Уравнение (Число A Число B Число C)
2         Число [a] 'x²' Число [b] 'x' Число [c] '=' '0'
3               {A=a, B=b, C=c}
4   () ()
5         'Решить' Уравнение {
6               Число D; D = B²-4*A*C;
7               Если D ≥ 0, то
8                     Печать (-B-√D)/(2*A);
9                     Печать (-B+√D)/(2*A).
10        }
11  < Решить -2,3x²+7x-1,2·10² = 0 >
```

Для получения недостающих языковых конструкций необходимо подгрузить знания, описанные на листинге 2, где объявлены и определены понятия Булево (строки 1-2, 13-14) и Число (строки 3-12), а также доопределено пустое понятие (строки 15-20).

Понятие Булево является простым, т.е. не содержит обобщаемых и ассоциируемых понятий. Для выражения этого понятия используются терминальные строки «Да» или «Нет», заданные шаблоном на языке регулярных выражений, заключаемым в кавычки (строка 2). Для упрощения примера семантика этой конструкции опущена.

В свою очередь понятие Число является обобщение понятия Булево, т.е. любая форма выражения понятия Булево является и формой выражения понятия Число. Однако, для преобразования сущности понятия Булево в сущность понятия Число, потребовалось ввести специальную языковую конструкцию (строка 4).

Основная форма выражения понятия Число задается регулярным выражением (строка 5). Для задания порядка интерпретации математических выражений вводится конструкция с круглыми скобками (строка 6) с пустой семантикой. Остальные языковые конструкции (строки 7-12) описывают основные математические операции над числами: модуль, квадратный корень и квадрат числа, унарный плюс и минус, бинарные операции сложения, вычитания, умножения и деления. Как и ранее, для упрощения примера семантика этих конструкций не указана.

Листинг 2 – Подгружаемые знания

```
1   () Булево ()
2         “Да|Нет” {…}
3   (Булево) Число ()
4         Булево {…}
5         “[0-9]+(,[0-9]+(·10[-+]?[0-9]+)?)?” {…}
6         ‘(’ Число ‘)’ {}
7         ‘|’ Число ‘|’ {…}
8         ‘√’ Число {…}
9         Число ‘2’ {}
10        “-|+” Число {…}
11        Число “-|+” Число {…}
12        Число “*|/” Число {…}
13  () Булево ()
14        Число “>|<|=|≠|≤|≥” Число {…}
15  () ()
16        ‘;’ {}
17        ‘Если’ Булево ‘,’ ‘то’ ‘’ ‘.’ {…}
18        ‘Число’ “[A-Za-zA-Яа-я][A-Za-zA-Яа-я0-9]*” {…}
19        Число ‘=’ Число {…}
20        ‘Печать’ Число {…}
```

Для внешней интерпретации знаний используется аксиома протоязыка, заключающаяся в том, что распознанное текстовое выражение понятия или терминальная строка, после которых указаны пустые квадратные скобки, передается текущему внешнему интерпретатору. Пример низкоуровневого описания семантики командами некоторой абстрактной виртуальной машины, приведен на листинге 3.

Листинг 3 – Низкоуровневая семантика

```
1   () ()
2         ‘#’ "[0-9]{3}" []{}     #\ Запись байта в область кода
3   () Число ()
4   ‘|’ Число ‘|’ {
5         #tst_                   #\ код 129, установки флагов операнда
6         #jns_ #001              #\ код 147, переход, если положительное
7         #neg_                   #\ код 131, изменение знака операнда
8         #label_ #001            #\ код 007, метка 001
9   }
```

На листинге 4 показан пример использования в качестве внешнего интерпретатора компилятора языка С.

Листинг 4 – Высокоуровневая семантика

```
1   () ()
2         `[' ".+" [] `]' {}      #\ Передача тела функции компилятору С
3   () Число ()
4         `|' Число [a] `|' {
5               [ (double a) { if( a ) return a; else return -a; } ]
6         }
```

Так как известные методы грамматического разбора оказались малопригодными для анализа текстов, порожденных контекстными и неоднозначными грамматиками, то для лексического, синтаксического и семантического анализа текстов в контекстной технологии применяется специально разработанный метод разнесенного грамматического разбора [24].

Таким образом, в контекстной технологии для решения каждой задачи и в процессе ее решения создается проблемный язык, содержащий понятийную структуру рассматриваемой проблемной области. Для этого выявленные в процессе понятийного анализа понятия включаются во множество понятий проблемного языка, а способы выражения понятий в тексте оформляются в виде языковых конструкций, которые и образуют его синтаксис. Описание семантики осуществляется методом математической индукции, заключающимся в использовании семантических категорий, которые определяются по мере необходимости, в процессе описания проблемного языка и средствами этого языка.

Основной недостаток контекстной технологии заключается в том, что коренным образом изменяется сама природа программирования. Вместо традиционного решения задач на языках общего применения, или на создаваемых для целого класса таких задач предметно-ориентированных языках, требуется отказаться от использования языкового подхода вообще, выполнить по определенным правилам описание привлекаемых предметных знаний (или использовать уже готовое такое описание) и выразить решение стоящей трудноформализуемой задачи в рамках описанных знаний.

Другим недостатком контекстной технологии программирования является эволюционность процесса программирования: при формализации знаний о проблемной области любой новый фрагмент знаний может быть определен только через те фрагменты, которые описаны ранее. В итоге решение любой задачи осуществляется «снизу вверх», от простого к сложному, от известного к неизвестному, от старого к новому, причем в основании этой пирамиды лежит аксиома протоязыка. Поэтому любую программу достаточно сложно написать, не опираясь на большие куски ранее определенных знаний.

6. Заключение

Эффективное решение трудноформализуемых задач видится в приближении выразительных средств языков программирования к индивидуальным проблемно-ориентированным языкам, свойственным каждой решаемой задаче. В этом случае понятия проблемно-ориентированного языка и способы их выражения в тексте программ совпадают или очень близки понятиям и способам их выражения в проблемной области трудноформализуемой задачи. Ни одна из известных технологий программирования, кроме контекстной, эффективными средствами для описания прикладных знаний не

располагает. Следует также заметить, что в настоящее время недоступна ни одна среда контекстного программирования, в которой можно было бы проверить возможности, декларируемые в описании контекстной технологии.

Список литературы

1. Одинцов И.О. Профессиональное программирование. Системный подход. 2-е изд. СПб.: БХВ-Петербург, 2004. 624 с.
2. Непейвода Н.Н., Скопин И.Н. Основания программирования. М.: РХД, 2003.
3. Выхованец В.С. Контекстная технология программирования // Труды IV Международной конференции «Параллельные вычисления и задачи управления» PACO '2008. Москва, 27-29 октября 2008 г. М.: Институт проблем управления им. В.А. Трапезникова РАН, 2008. С. 1288-1327.
4. Выхованец В.С., Говоров М.И. Трудноформализуемые задачи и контекстная технология программирвоания // Материалы VII Международной конференции «Управление развитием крупномасштабных систем» MLSD '2013. Москва, 30 сентября - 2 ноября 2013 г. М.: Институт проблем управления им. В.А. Трапезникова РАН, 2013. Т. 2. С. 315-318.
5. Формальные спецификации в технологиях обратной инженерии и верификации программ / Бурдонов И.Б., Демаков А.В., Косачев А.С. и др. // Труды Института системного программирования. 1999. № 1. С. 31-43.
6. Фаулер М. Предметно-ориентированные языки программирования. М.: Вильямс, 2011.
7. Грофф Д.Р., Вайнберг П.Н., Оппель Э.Дж. SQL: полный справочник. 3-е изд. М.: Вильямс, 2011. 960 с.
8. Фридл Дж. Регулярные выражения. СПб.: Питер, 2001. 352 с.
9. XML Path Language (XPath) 3.0 [Электронный ресурс]. Режим доступа: http://www.w3.org/TR/xpath-30/
10. Таненбаум Э.С., Вудхалл А.С. Операционные системы. Разработка и реализация. 3-е изд. СПб.: Питер, 2007. 704 с.
11. Соломатин Д.И. Синтаксически расширяемые языки программирования как средство реализации языков предметной области // Вестник ВГУ. Системный анализ и информационные технологии. 2008. № 1. С. 142-150.
12. Nemerle. Programming language for "special forces" of developers [Электронный ресурс]. Режим доступа: http://nemerle.org/About.
13. Фаулер М. Языковой инструментарий: новая жизнь языков предметной области [Электронный ресурс]. 2005. Режим доступа: http://www.maxkir.com/sd/languageWorkbenches.html.
14. Дмитриев С. Языково-ориентированное программирование: следующая парадигма [Электронный ресурс] // RSDN Magazine. 2005. №5. Режим доступа: http://www.rsdn.ru/article/philosophy/LOP.xml
15. Ларман К. Применение UML 2.0 и шаблонов проектирования. 3-е изд. М.: Вильямс, 2006. 736 с.
16. Levine R., Mason T., Brown D. LEX & YACC. Sebastopol: O'Reilly & Associates, 1992.
17. Donnelly C., Stallman R. Bison: YACC-compatible parser generator. Boston: Free Software Foundation, 1995.
18. Meta Programming System. DSL Development Environment [Электронный ресурс]. Режим доступа: http://www.jetbrains.com/mps/
19. Software Factories [Электронный ресурс]. Режим доступа: http://msdn.microsoft.com/en-us/library/ff699235.aspx
20. Intentional Workbench [Электронный ресурс]. Режим доступа: http://www.intentsoft.com
21. Выхованец В.С. Методы анализа крупномасштабного производства. Понятийный анализ и моделирование // Труды III Международной конференции «Управление развитием крупномасштабных систем» MLSD '2009. М.: Институт проблем управления им. В.А. Трапезникова РАН, 2009. С. 308-316.
22. Выхованец В.С. Описание семантики контекстно-свободных языков методом математической индукции // НТИ. Сер. 2. Информационные процессы и системы. 2008. № 7. С. 6-14.
23. Выхованец В.С., Иосенкин В.Я. Понятийный анализ и контекстная технология программирования // Проблемы управления. 2005. № 4. С. 2-11.
24. Выхованец В.С. Разнесенный грамматический разбор // Проблемы управления. 2006. № 3. С. 32-43.