Санкт-Петербургский Государственный Университет

Математико-Механический факультет

Кафедра системного программирования

Разработка языка представления знаний C#-Expert

Дипломная работа студента 59-А группы

Новикова Антона Владимировича

Научный руководитель
(профессор, доктор технических наук)

/подпись/_____/Сафонов Владимир Олегович/

Рецензент
(кандидат технических наук)

/подпись/_____/Глазунов Александр Григорьевич/

“Допустить к защите”
(заведующий кафедрой, доктор физико-математических наук)

/подпись/_____/Терехов Андрей Николаевич/

Санкт-Петербург

2004 год

Оглавление

Введение

Целью данной дипломной работы является создание языка представления знаний C# Expert, который может использоваться как основа технологии разработки экспертных систем. В настоящее время существует множество различных экспертных систем (ЭС) таких как VP Expert, KEE, GURU и другие. В чем особенность C# Expert?

Базовым языком C# Expert выбран объектно-ориентированный язык C#. Такое решение дает возможность применять при создании ЭС объектно-ориентированный подход, привычный для большинства программистов, а также использовать существующие библиотеки, разработанные под платформу .NET;

C# Expert представляет собой расширение языка C#. Благодаря этому исходная программа может содержать как код специфичный для экспертных систем (фреймы, слоты и др.), так и простой код на языке C#.

В качестве прототипа языка C# Expert, был выбран язык Турбо Эксперт созданный проф. Сафоновым В.О.. На мой взгляд, возможности Турбо Эксперта выгодно отличают данный язык от других базовых языков ЭС, в нем выделено несколько способов представления знаний каждая из которых играет определенную роль; фреймы служат для описания структуры и иерархии объектов, для управления рассуждениями и организации работы системы в целом; наборы правил реализуют логический вывод; процедурные знания (процедуры и модели) описывают вычислительные алгоритмы и другие в традиционном смысле алгоритмические элементы знаний. Также в отличие, например, от ЭС GURU, в Турбо Эксперте возможно использование сложных типов в описании слотов фреймов, множественное наследование фреймов и многое другое.

Основные возможности языка C# Expert:

i. Возможность представления знаний о структуре, иерархии и классификации понятий и объектов;

ii. Возможность взаимодействия с крупными моделями алгоритмического знания – программами и компонентами написанными для платформы .NET;
iii. Возможность представления нечетких знаний;
iv. Возможность описания и использования наборов правил (продукций);

В ходе выполнения диплома, проделаны следующие работы:

i. Разработан синтаксис и семантика языка C# Expert;
ii. Описана грамматика языка с помощью пакета для создания компиляторов CoCo/R;
iii. Реализован рабочий прототип метакомпилятора C# Expert с поддержкой основной функциональности языка;
iv. Реализовано несколько несложных примеров на C# Expert, которые демонстрируют общие принципы использования языка;

Глава 1. Обзор существующих экспертных систем

Экспертная система (ЭС) – это программная система, осуществляющая экспертную деятельность при решении конкретной задачи в определенной предметной области (медицина, химия, экономика и др.), используя базу знаний (БЗ) для получения решений.
Некоторые примеры экспертных систем:

i. Сеть американских магазинов Wall-Mart использует экспертную систему, которая собирает в реальном времени информацию о продажах во всех магазинах торговой сети (всего 3000 магазинов), анализируя полученные данные, ЭС устанавливает зависимости продаж различных товаров, позволяя тем самым предсказывать продажи каждого вида продукта в каждом магазине сети с большой точностью, что, в итоге, приводит к оптимизации затрат.
ii. Медицинская экспертная система FocalPoint, разработанная компанией TriPath, изучает анализы (мазок Папаниколау) пациентов, позволяя обнаруживать раковые заболевания с хорошей степенью достоверности.

Данная система просматривает огромное количество анализов ежегодно, более 5 миллионов, примерно 10% от всех сданных анализов на мазок Папаниколау в США, позволяя диагностировать заболевание на ранней стадии.

iii. Экспертная система PROSPECTOR предназначена для обнаружения месторождений полезных ископаемых. С помощью PROSPECTOR было открыто крупное месторождение молибдена в США.

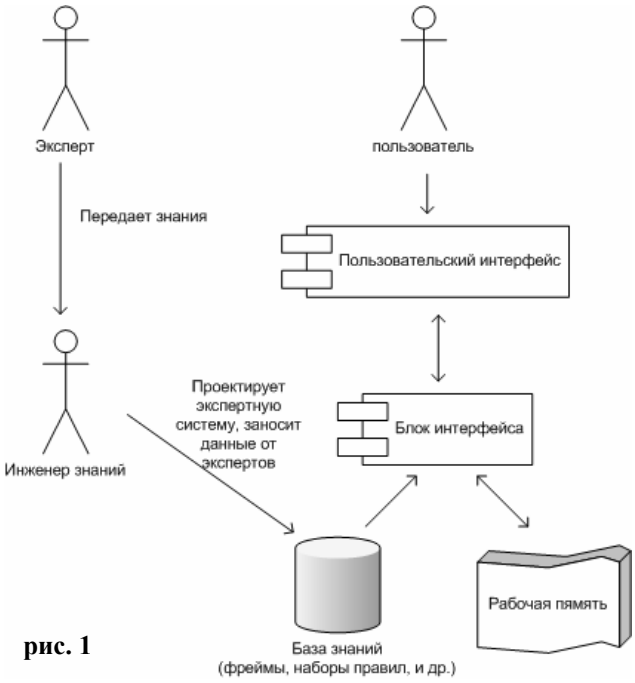


рис. 1

На мой взгляд, экспертные системы не только позволяют заменить обычных экспертов, но, и в некоторых областях, превосходят их по точности консультации. Можно выделить следующие основные преимущества ЭС по сравнению с обычным экспертом:

i. Быстрое обучение знаниям;
ii. Никогда не забывают ту информацию, которую изучили;
iii. Возможность делать выводы на основе огромного количества информации;
iv. Также как и обычные эксперты, ЭС могут принимать решения основываясь на недостаточных и/или нечетких данных;
v. Дешевы в эксплуатации (не надо платить за каждую консультацию);
vi. Всегда готовы консультировать (нет выходных, отпусков и т.п.);

Важной привлекательной чертой ЭС для пользователей, большинство из которых не имеет представления о программировании, является возможность работы в диалоговом режиме без использования какого-либо специального

алгоритмического языка. Характерной особенностью ЭС, в отличие от других классов программ (трансляторов, систем управления базами данных, пакетов прикладных программ и др.), также является ее способность объяснять пользователю ход своих рассуждений. Важную роль в создании и проектировании экспертных систем играют инженеры знаний.

Общая схема ЭС представлена на рисунке выше (рис. 1). Как видно из рисунка, основные задействованные роли это:

i. Эксперт в заданной предметной области;
ii. Инженер знаний;
iii. Пользователь экспертной системы;

Эксперт – это специалист в некоторой области, на основе его знаний инженер знаний проектирует базу знаний. Пользователь – это человек, использующий экспертную систему посредством получения консультации в заданной предметной области. Например, в случае экспертной системы PROSPECTOR, вероятным пользователем будет геолог, получающий консультацию у системы о вероятности наличия той или иной горной породы в заданной местности.

База знаний (БЗ) – компонент ЭС, инкапсулирующий в себе знания эксперта. Основные подходы к организации БЗ:

1. Фреймовая структура. Данные представляются в виде иерархического дерева фреймов, где каждый фрейм отражает некоторое понятие, объект или абстракцию.
2. Наборы правил. Данные представляются в виде правил (продукций) if-then-else. Экспертная деятельность в таких системах обычно осуществляется через машину логического вывода.

Базовый язык ЭС – язык для описания БЗ экспертной системы.

Глава 2. Сравнительный анализ C# Expert и других базовых языков ЭС

2.1. Выбор инструментальной среды разработки

Большинство существующих экспертных систем используют в качестве базовых языков Prolog и Lisp. Такой подход мотивируется удобством использования данных языков в задачах искусственного интеллекта. На наш взгляд, такой выбор имеет ряд недостатков. Во-первых, нестандартность семантики этих языков требует специальной подготовки инженеров знаний. Например, на языке Prolog, выполнение программы это не последовательное исполнение команд, а вывод некоторой переменной исходя из начальных значений и правил, а в языке Lisp используется обратная запись для представления арифметических выражений. Соответственно, в этом случае, инженеры знаний должны иметь специальную подготовку для создания ЭС на базовых языках такого рода. Другой недостаток данного подхода, это плохая приспособленность этих языков к задачам разработки графического интерфейса. Удобный, интуитивно понятный графический интерфейс – это важная составляющая качественной экспертной системы, т.к. ЭС, прежде всего, ориентированны на обычных пользователей, без навыков программирования.

Язык представления знаний C# Expert основан на языке C#. Такое решение имеет ряд преимуществ. Прежде всего, при разработке ЭС на C# Expert'е может быть применен объектно-ориентированный подход привычный большинству программистов (инженеров знаний), более того, C# популярный язык, на котором легко могут работать также специалисты использующие C++ или Java. Все это делает возможным использование C# Expert широким кругом специалистов (инженеров знаний). C# – это язык платформы .NET. Таким образом, еще одно значительное преимущество выбранного подхода, это возможность разрабатывать ЭС под .NET. На сегодняшний день, .NET – это наиболее современная и перспективная платформа для коммерческих приложений. Существует множество уже готовых решений под .NET. Следует также отметить, что .NET обеспечивает хорошую межъязыковую совместимость и широкие возможности использования программных компонент (Assembly, COM, DLL).

2.2. Принципы построения языка и системы C# Expert

Основная сложность при создании экспертных систем это представление знаний экспертов в базе знаний наиболее подходящим образом для решения задач в заданной области. Для обеспечения такой возможности, базовый язык ЭС должен иметь четкий, хорошо структурированный способ представления данных и знаний.

К примеру, хороший способ представления знаний обеспечивает ЭС GURU разработанная фирмой Micro Data Base Systems, США. Эта система ориентирована на разработку ЭС в области деловых расчетов. К полезным возможностям GURU следует отнести возможность описания массивов как элементов данных, поддержка работы с таблицами и базами данных. Более того, в отличие от многих других ЭС, GURU предоставляет интегрированный подход к обработке данных, позволяя совместное использование наборов правил (продукций) с таблицами и реляционными базами данных. В ЭС GURU реализована гибкая подсистема логического вывода, позволяющая осуществлять как прямой, так и обратный вывод на одних и тех же наборах правил, имеющая способ разрешения конфликтов правил (conflict resolution) с помощью указания их приоритетов и порядка выполнения. Также GURU позволяет работать с нечеткими знаниями и использовать нечеткий логический вывод на основе коэффициентов уверенности, поддерживается возможность использования различных формул для вычисления коэффициента уверенности. Тем не менее, в системе GURU содержится и ряд ограничений, к примеру, отсутствуют средства представления сложных структурированных объектов и понятий сложной структуры данных, не предусмотрена возможность описания процедурных знаний. По-видимому, эти ограничения объясняются спецификой области применения: деловые расчеты, а также, возможно, устарелостью системы в целом.

В качестве, примера другой удачной экспертной системы, можно упомянуть ЭС KEE (Knowledge Engineering Environment). Это фреймовая экспертная система, где основным элементом данных базы знаний является юнит (фрейм). Юниты состоят из слотов, а слоты в свою очередь могут содержать данные простых типов (число, строка и т.п.), таблицы, графику, указатели на другие юниты или

процедурные знания, написанные на языке Lisp. В системе KEE также реализован механизм наследования, который позволяет организовывать юниты в иерархические структуры, обеспечивая логически связанное представление информации в базе знаний. Безусловно, фреймовая структура данных, реализованная в системе KEE, обеспечивает более широкие возможности представления данных, чем структуры данных ЭС GURU. Основным недостатком ЭС KEE является использование языка Lisp в качестве базового языка системы, и как следствие сложная семантика базового языка с достаточно нетрадиционной формой записи для большинства инженеров знаний.

Принимая во внимания опыт вышеперечисленных экспертных систем, сформулируем основные требования к языку представления знаний C# Expert:

i. Возможность представления знаний о структуре, иерархии и классификации понятий и объектов.
ii. Возможность взаимодействия с крупными существующими программными компонентами (компонентами алгоритмических знаний)
iii. Возможность интеграции с графическим интерфейсом;
iv. Возможность представления нечетких знаний;
v. Возможность описания и использования наборов правил;
vi. Возможность применения различных машин логического вывода;

Отметим, что фреймовая организация C# Expert близка к структуре юнитов в экспертной системе KEE, а в качестве прототипа организации логического вывода используется ЭС GURU.

2.3. Изобразительные средства базового языка

Изобразительные возможности, на наш взгляд, являются ключевой характеристикой базового языка. Проблема базовых языков многих экспертных систем – это либо ограниченность выразительных возможностей в смысле традиционного программирования, т.е. отсутствие сложных структур данных и средств типизации, например, VP-EXPERT, GURU, либо не привычный синтаксис и

семантика базового языка (ЭС основанные на прологе или языке Lisp).

Сформулируем основные требования к базовому языку:

i. Должен обладать привычным синтаксисом и семантикой для большинства инженеров знаний.
ii. Имеет богатый набор средств структурирования данных и программы.

Таким образом, базовый язык экспертной системы должен сочетать в себе черты языка представления знаний и языка программирования. Исходя из вышеперечисленных рассуждений, нам представляется целесообразным использовать в качестве базового языка экспертной системы расширение одного из популярных языков программирования. По ряду причин, перечисленных выше (см. раздел 2.1), в данной дипломной работе выбран язык C#.

Глава 3. Описание разработанного языка C# Expert

3.1. Способы представления знаний

C# Expert – это фреймовый язык программирования включающий в себя возможность использования правил и машины логического вывода (см. раздел 2.2) таким образом основные структурные элементы используемые для представления знаний это:

i. Фреймы (фреймы-классы, фреймы-экземпляры, фреймы-наборы правил);
ii. Слоты (instance slot, own slot);
iii. Правила;

Также, поскольку C# Expert является расширением языка C#, можно использовать для представления знаний структурные элементы языка C# (классы, пакеты, библиотеки), а также подключать внешние модули (Assembly, DLL, COM компоненты) средствами языка C#.

3.2. Общая структура программы на C# Expert

Программа написанная на C# Expert состоит из двух частей:

i. Исходный код на C#;
ii. Исходный код специфичный C# Expert (фреймы, слоты и т.д.);

Исходный код на C# разделяется с исходным кодом на C# Expert с помощью ключевого слова *“#frames”*.

Из исходного кода на C# имеется возможность использовать фреймы и их атрибуты и их свойства (атрибуты и слоты) используя стандартный способ адресации *[имя_фрейма].[свойство]*.

Листинг 1

```
using System;

// C# native code
namespace HelloWorld
{
    class Hello
    {
        [STAThread]
        static void Main(string[] args)
        {
             Console.out.WriteLine (“Value: {0}”, Test.title);
        }
    }
}

// C# Expert specific code
#frames
frame class Test
{
    own_slots
       string title=”Hello world!”;
}
```

3.3. Фреймы

Структура фрейма

```
frame FR_CATEGORY FR_IDEN {
       [ATTRIBUTES]
       SlOTS
}
FR_CATEGORY – категория фрейма может быть: class, instance,
ruleset;
FR_IDEN – идентификатор фрейма, уникальный в пределах программы;
ATTRIBUTES – атрибуты фрейма (необязательные);
SLOTS – слоты фрейма;
```

Фрейм – концепция, используемая для представления в базе знаний различного рода абстракций (объекты, понятия) и их организации с помощью иерархической структуры. Основные свойства поддерживаемых фреймов:

i. Множественное наследование;
ii. Встроенные средства хранения данных во внешней памяти. Объекты языка C#, как и обычные переменные, носят оперативный характер, т.е. их значения не сохраняются от запуска к запуску программы.

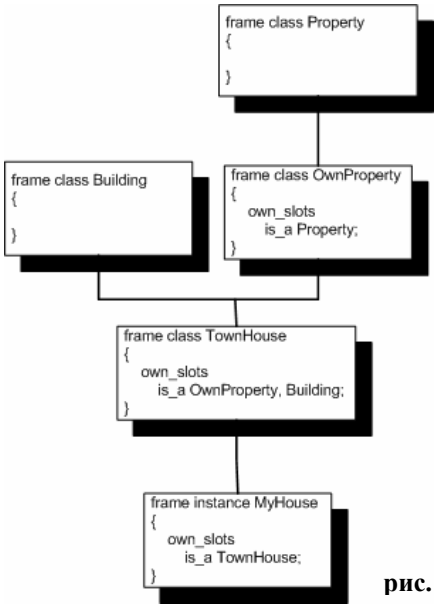


рис. 2

Поддерживаются фреймы следующих видов:

i. Фрейм-класс;
ii. Фрейм-экземпляр;
iii. Фрейм-набор правил (см. раздел 3.4);

Фрейм-классы используются для описания классов объектов, а фрейм-экземпляры для конкретных экземпляров некоторого класса. Используются правила наследования (рис. 2):

i. Фрейм-экземпляр должен быть унаследован от фрейм-класса (множественное наследование не допускается[1]);
ii. Фрейм-класс может быть унаследован от нескольких других фрейм-классов (допускается множественное наследование);

Также на фреймы накладываются следующие ограничения:

i. Не разрешается определять фрейм во фрейме, все фреймы должны быть определены в глобальном пространстве программы;
ii. Фреймы не могут создаваться динамически с помощью оператора new;

Фрейм состоит из структурных единиц двух типов:

i. Атрибуты;
ii. Слоты;

Атрибуты

Атрибуты – любой код на языке C#, может содержать объявления классов, делегатов, переменных, констант, методов, ссылок на делегаты. Объявление любого атрибута может содержать дескриптор указывающий область видимости: private, protected, public. По умолчанию, используется дескриптор “private”. Атрибуты фрейма не сохраняются во внешней памяти, т.е. их значения при перезапуске программы обнуляются.

```
ATTRIBUTES ::=
 C_SHARP_CODE

C_SHARP_CODE – любой код на языке C#.
```

[1] Описание см. в параграфе «Слоты»

Слоты

Слоты – структурные элементы, описывающие свойства фрейма. Как и любые переменные C#, значения слотов могут быть простыми и сложными типами, в том числе могут быть делегатами. Основные отличия слотов от переменных C#:

i. Значения слотов сохраняются во внешней памяти между запусками экспертной системы.
ii. Возможность описания «присоединенных процедур» («демонов»), то есть процедур вызываемых автоматически при определении, модификации или использовании слотов.
iii. Слоты могут динамически добавляться и удаляться из фрейма в отличие от обычных свойств класса (атрибутов).

```
SLOTS ::=
  own_slots
    [[SL_DESC] SL_NAME[=SL_VALUE];]+
  [
  instance_slots
    [SL_DESC SL_NAME[=SL_VALUE];]+
  ]

SL_DESC – описание (тип) слота.
SL_NAME – имя слота, уникальное в пределах фрейма.
SL_VALUE – значение слота, в случае слота экземпляра (instance
slots), это значение по умолчанию
```

Расширенное объявление слота доступно через фасеты:

```
facets {
       type TYPE;
       [value VALUE;]
       [default_value DEF_VALUE;]
       [restriction B1]
       [if_needed B2]
       [if_added B3]
       [if_modified B4]
       [if_removed B5]

} SL_NAME;

TYPE – описание (тип) слота
VALUE – значение слота, м.б. не определено для own_slots если
определено default_value в порождающем фрейм-классе;
DEF_VALUE – позволяет задать значение по умолчанию для
instance_slots во фрейм-классе;
B1 – код на C#, задает ограничения значения слота;
B2, B3, B4, B5  – код на C#;
```

При расширенном объявлении слота (через фасеты) возможно задание ограничений на значения слота. Код ограничений будет вызываться при каждом изменении значения слота, при этом переменная current_value будет содержать текущее значение слота, а переменная new_value новое значение, допустимость которого проверяется (см. пример ниже).

```
restriction
{
    if (new_value>100)
    {
        current_value = 100;
    } else if (new_value<0)
    {
         current_value = 0;
    } else
    {
         current_value = new_value;
    }
}
```

Поддерживаются следующие виды присоединенных процедур:

i. if_needed – вызывается при каждом обращении к слоту. Вызывается перед получением значения;
ii. if_added – вызывается при определении значения данного слота. Вызывается сразу после определения значения слота;
iii. if_modified – вызывается при изменении значения данного слота. Вызывается сразу после изменения значения слота;
iv. if_removed – вызывается при удалении данного слота. Вызывается сразу после удаления значения слота;

Фрейм-класс

Фрейм-классы предназначены для описания классов каких-либо объектов или понятий предметной области экспертной системы. Фрейм-класс может быть унаследован от одного или более других фрейм-классов (допустимо множественное наследование).

```
frame class FR_IDEN {
     [ATTRIBUTES]
[own_slots
     [is_a FR_IDEN1; | FR_IDEN2, FR_IDEN3, FR_IDEN4, …]
```

```
     [SLOTS]
]
[instance_slots
     [SLOTS]
]
```

FR_IDEN1, FR_IDEN2, … – идентификаторы фрейм-классов (надклассов) от которых унаследован данный фрейм-класс;
own_slots – содержит описание слотов, принадлежащих классу;
instance_slots – содержит описание слотов, принадлежащих экземплярам унаследованным от данного класса;

В текущей реализации C# Expert, имена слотов среди фрейм-класса и его надклассов должны быть уникальными.

Фрейм-класс может содержать два различных вида слотов:

i. Собственные слоты (own slots);
ii. Слоты экземпляров (instance slots);

Все слоты экземпляров (instance slots), описанные во фрейм-классе, наследуются фрейм-экземплярами данного класса как свои собственные слоты, при этом одни и те же унаследованные слоты могут иметь у разных экземпляров разные значения. Кроме того, фрейм-экземпляр может иметь и дополнительные слоты, характеризующие его индивидуальные свойства. Таким образом, у разных фрейм-экземпляров одного и того же фрейм-класса набор и значения собственных слотов может различаться. В случае если унаследованный слот экземпляра (instance slot) не переопределяется во фрейм-экземпляре, используется значение по умолчанию определенное во фрейм-классе.
Значение по умолчанию определяется:

i. при использовании короткой формы записи через оператор присваивания:

```
SlotType1 slot1 = defaultValue;
```

ii. В случае расширенной записи, значение по умолчанию определяется через фасет default_value:

```
facets
{
    type SlotType1;
```

```
        default_value defaultValue;
    }slot1;
```

Фрейм-экземпляр

Фрейм-экземпляры используются для описания объектов некоторого класса объектов предметной области экспертной системы.

```
frame instance FR_IDEN {
    [ATTRIBUTES]
own_slots
    SLOTS
is_a FR_IDEN1;
}
```

Собственные слоты фрейм-экземпляра, которые унаследованы от фрейм-класса, инициализируются без указания типа, например:

```
slotName = slotValue;
```

или

```
facets
{
    value: slotValue;
} slotName;
```

3.4. Наборы правил

Набор правил задает совокупность продукций используемых для логического вывода. В C# Expert, для унификации синтаксиса, наборы правил рассматриваются как тип фрейма. Как и другие фреймы, наборы правил содержат атрибуты и слоты. Поддерживаются следующие типы слотов:

i. правила составляющие набор;
ii. целевая переменная;
iii. контекст фреймов в котором набор правил исполняется;

Форма представления набора правил близка к принятой в системе KEE.

Основные отличия концепции набора правил в языке Турбо-Эксперт от наборов правил используемых в других экспертных системах:

i. Возможность описания помимо правил атрибутов, как и во фреймах других видов;
ii. Возможность параметризации набора правил, аналогично параметризации функции;
iii. Возможность указания конкретных экземпляров фреймов-классов, в контексте которых выполняется набор правил;
iv. Возможность указания конкретных фреймов-экземпляров в которых выполняется набор логический вывод;
v. Возможность изменения стратегии вывода и переопределение машины вывода.

Все вышеперечисленные возможности придают процессу логического вывода значительно большую гибкость, чем в распространенных продукционных оболочках таких как GURU и VP-EXPERT.

Описание набора правил имеет вид:

```
frame ruleset RS_IDEN [([ref]TYPE1 PARAM1, [ref]TYPE2 PARAM2, …)]
{
[ATTRIBUTES]
own_slots
   RULESET_SLOTS
}
ATTRIBUTES – любой C# code (также как и атрибуты во фреймах других
типов)
RS_IDEN – идентификатор фрейма
ref – ключевое слово, задает параметры-переменные;
TYPEx – тип параметра, может быть любой тип C#;
PARAMx – имя параметра;
RULESET_SLOTS – слоты набора правил;
```

Как и для функций в набор правил допускается передавать параметры по значению и по ссылке. В программе C# инициализация параметров набора правил RS_IDEN осуществляется с помощью вызова метода (порядок параметров должен быть такой же как в описании набора правил):

```
RS_IDEN.initParameters([ref] TYPE1 PARAM1, [ref]TYPE2 PARAM2, …);
```

Параметры, которые были переданы по ссылке, можно получить с помощью метода:

```
RS_IDEN.getParameters(ref TYPE1 PARAM1, ref TYPE2 PARAM2, …);
```

Рассмотрим описание набора правил:

```
RULESET_SLOTS::=
       context [instance]F1, [instance]F2, … |
       goal V;|
       [rule R {
             if (B1)
             then  S2;
             [else S3;]
        [comment "Some text";]
        [priority INT_VALUE;]
       }]+
```

i. context [instance]**F1**, [instance]**F2**, … | – данный слот задает совокупность имен фрейм-классов и фрейм-экземпляров в контексте которых работает данный набор правил. Если F1 имя фрейма класса, то в контексте набора правил могут быть использованы любые его собственные слоты (own slots), Если перед именем класса задано ключевое слово instance, то фрейм может работать в контексте любого фрейм-экземпляра унаследованного от этого класса. В этом случае в наборе правил могут использовать собственные слоты и слоты экземпляра. Фрейм-экземпляр передается по имени с помощью метода initContext():

```
RS_IDEN.initContext(string F1, string F2, …);
```

Текущая реализация C# Expert не позволяет использовать в контексте фреймы, которые имеют слоты с одинаковыми именами.

goal **V; –** задает целевую переменную набора правил используемую по умолчанию. Целевая переменная может быть переопределена при вызове метода consult();

ii. Описание правила:

```
rule R
{
     if (B1)
     then S2;
     [else S3;]
}
R – имя правила;
B1 – условие правила, представляется в виде выражения
```

```
принимающего значения истинна(true) или ложь(false);
S2 – заключение (код C# исполняемый в случае
истинности условия[2]);
S3 – код C# исполняемый в случае если выражение B2
ложно;
```

iii. comment “Some text”; – позволяет задать комментарий отображаемый в случае вызова правила в ходе консультации. Машина логического вывода поставляемая вместе с библиотекой «C# ExpertLibrary» позволяет включать и отключать отображение комментариев с помощью свойства bool ProductionSystem.showComments;

iv. priority **INT_VALUE**; – позволяет определить приоритет правила, данный слот может использоваться машиной логического вывода для разрешения конфликтов (в случае если на каком-либо шаге вывода несколько правил имеют истинное значение условия). Отметим, что машина вывода ProductionSystem, поставляемая вместе с библиотекой CSharpExpertLibrary не использует приоритеты.

Глава 4. Описание реализации языка C# Expert

4.1. Общая схема использования

Как говорилось выше, C# Expert является расширением языка C#. Программа написанная на C# Expert конвертируется с помощью специального метакомпилятора (конвертора) в программу на C#. Получившаяся программа, в свою очередь, компилируются компилятором C# в запускаемый файл платформы .NET. При последней стадии используется библиотека CSharpExpertLibrary.dll поставляемая вместе с метакомпилятором. Общая схема использования показана на рисунке ниже (рис. 3)

[2] Необходимо отметить, что код S2 не всегда исполняется в случае истинности условия B1, в действительности, зависимость B1, S2 и S3 между собой зависит от реализации машины логического вывода

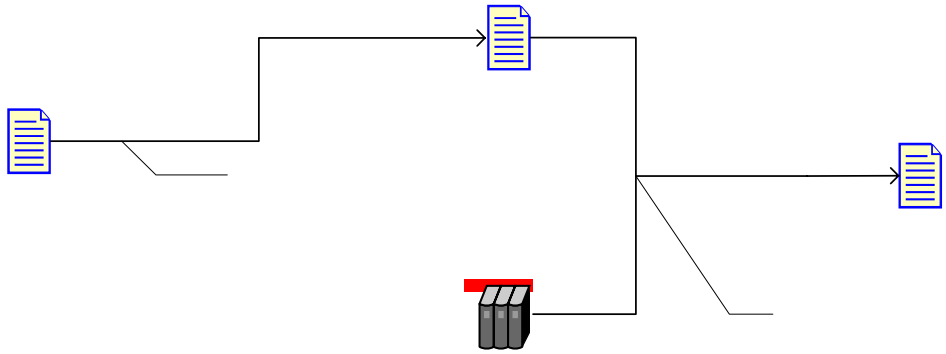


рис. 3

Для использования C# Expert метакомпилятора и библиотеки необходимо иметь на компьютере следующие приложения:

i. .NET Framework 1.1;
ii. Microsoft Development Environment 2003;

Программа на C# Exp

4.2. Метакомпилятор (конвертор) языка C# Expert

Задача метакомпилятора (конвертора) C# Expert транслировать программы, написанные на языке C# Expert (описание языка см. Глава 3), в программы на языке C#. Общая схема работы конвертора:

Sample.expert

i. Синтаксический разбор программы на C# Expert и построение промежуточного кода. Синтаксический анализатор написан с помощью инструмента построения компиляторов «CoCo/R Compiler Generator». Используется грамматика C# расширенная конструкциями языка C# Expert. Описание грамматики см. в разделе «Приложения»;
ii. Генерация программы C# из промежуточного кода. Особое внимание уделяется замене в результирующей программе идентификаторов доступа к слотам фреймов. Необходимость такой замены связана с двумя причинами:
    a. Необходимость обеспечения возможности динамического удаления/добавления слотов во фрейме;

b. В программе на C# Expert обращаясь к слоту с помощью идентификатора типа frame.slot мы получаем его значение. В действительности же слот представляет из себя сложный объект содержащий помимо значения, другие свойства, такие как присоединенные функции, ограничения и значение по умолчанию (см. раздел 3.3 параграф “Слоты”);

Поэтому при генерации результирующего кода слоты фрейма представляются в виде объектов класса Slot и содержаться в коллекции CSharpExpertAbstract.slots.

Общая схема работы конвертора показана на рисунке ниже

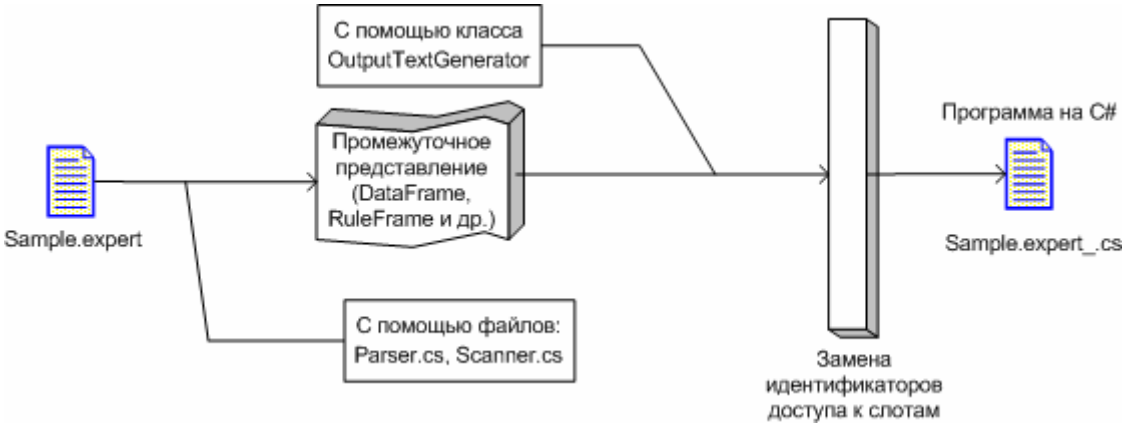


рис. 4

Точка входа в C# Expert конвертор находиться в методе ExpComp.Main(string[] arg); Метод получает имя исходного файла и запускает метод Parser.Parse() который осуществляет синтаксический анализ и генерацию промежуточного кода. После этого вызывается метод OutputTextGenerator.generateOutputProgram (StreamWriter s) который проводит семантический анализ промежуточного кода, замену идентификаторов доступа к слотам и генерацию результирующей программы на C#.

Описание основных классов метакомпилятора C# Expert

На этапе генерации промежуточного кода, основные функции выполняют классы Parser и Scanner сгенерированные инструментом для создания компиляторов CoCo/R. В статические структуры класса ExpComp добавляются объекты образующие промежуточное представление:

i. Фреймы добавляются в коллекцию IDictionary ExpComp.dataFrames;
ii. Наборы правил добавляются в коллекцию IDictionary ExpComp.ruleFrames
iii. Все идентификаторы используемые в исходном коде добавляются в таблицу идентификаторов VarTable ExpComp.varTable. UML диаграмму промежуточного кода см. в разделе «Приложение 2».

Генерацию результирующего файла на C# из промежуточного кода осуществляет класс OutputTextGenerator. Рассмотрим основные методы этого класса:

i. private static string OutputTextGenerator.changeSlotAccessors (CScode code, …) — заменяет в заданном фрагменте кода идентификаторы доступа к слотам;
ii. public void generateOutputProgram (StreamWriter s) – проводит семантический анализ объектов промежуточного представления кода и создает результирующую программу на C#;

Семантический анализ осуществляется с помощью метода public bool ISemanticsChecker.checkSemantics(). Основные классы промежуточного представления, такие как DataFrame, RuleFrame имплементируют интерфейс ISemanticsChecker.

4.3. Библиотека языка C# Expert

Как описывалось выше, метакомпилятор конвертирует программу на C# Expert в программу на языке C# которая использует библиотеку CSharpExpertLibrary. Данный раздел содержит описание принципов построения библиотеки. Основным классом библиотеки является CSharpExpertAbstract. Он состоит из статических атрибутов и методов, которые предоставляют интерфейс для доступа к фреймам и наборам правил, позволяют сохранить и загрузить данные в/из базы знаний. При конвертации программы из C# Expert в C# метакомпилятор создает класс CSharpExpert унаследованный от класса CSharpExpertAbstract.

Общая схема взаимодействия объектов библиотеки показана на рисунке ниже (рис. 5):

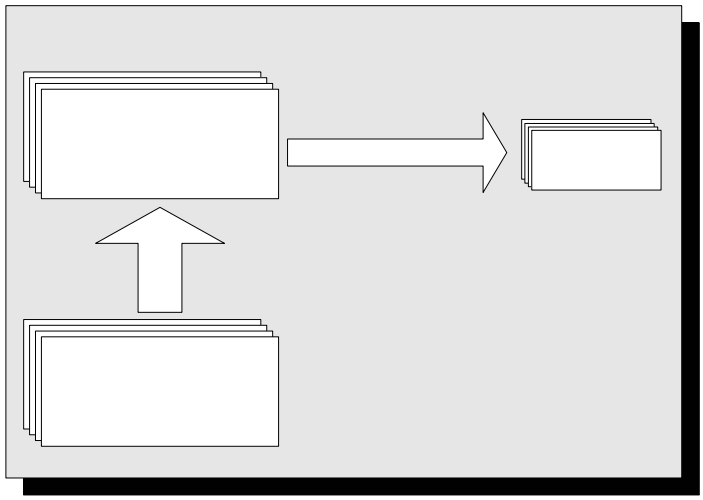


рис. 5

Список понятий языка C# Expert и их имплементации в библиотеке CSharpExpertLibrary:

i. Фрейм-класс – ClassFrame;
ii. Фрейм-экземпляр – InstanceFrame;
iii. Набор правил – RulesetFrame;
iv. Слот – Slot;
v. Правило – Rule;

В разделе приложений присутствует UML диаграмма классов библиотеки (см. раздел “Приложение 3”).

Ниже приводиться подробное описание классов библиотеки которые отражают структуры языка C# Expert.

Abstract Class CSharpExpertAbstract

Атрибуты

i. `[NonSerialized]`
`protected IDictionary dataFrames`
Содержит все «фреймы-экземпляры» и «фреймы-классы» объявленные в системе.
Ключ – это идентификатор (имя) фрейма, значение – экземпляр класса, унаследованного от абстрактного класса ClassFrame или InstanceFrame в зависимости от типа фрейма.

ii. `[NonSerialized]`
`protected IDictionary rulesetFrames`
Содержит все «фреймы-наборы правил» объявленные в системе. Ключ – это идентификатор (имя) фрейма, значение – экземпляр класса, унаследованного от абстрактного класса RulesetFrame. Класс RulesetFrame представляет в системе сущность frame ruleset.

iii. `protected IDictionary slots`
Содержит все слоты объявленные во фреймах. Ключ – это полностью определенный (full qualified) идентификатор слота. Например, слот age фрейма Elephant будет иметь полностью определенный идентификатор равный Elephant.age; В отличие от dataFrames и rulesetFrames коллекций, данная коллекция сериализуема. Сериализация позволяет сохранять базу знаний между запусками экспертной системы.

iv. `protected static CSharpExpertAbstract instance`
Содержит экземпляр унаследованного класса CSharpExpert. Экземпляр создается с помощью метода CSharpExpert.createInstance().

Статические методы

i. `public static void CSharpExpert::loadKnoweldgeBase(string fname)`
загружает базу знаний из файла. Сохранение и восстановление базы знаний реализовано с помощью сериализации классов C#.
Параметры: string fname – имя файла из которого загружается база знаний, если имя файла не задано то по умолчанию используется имя «CSharpExpertKnowledgeStorage.bin»

ii. `public static void CSharpExpert::saveKnoweldgeBase(string fname)`
сохраняет базу знаний в файл. Сохранение и восстановление базы знаний реализовано с помощью сериализации классов C#.

iii. `public static void CSharpExpert::firstStartAbstract()`
этот метод используется при первом старте системы, когда база знаний еще не создана. Создает новые экземпляры фрейм-классов и фрейм-экземпляров.

iv. `public static void addSlot (string iden, Slot slot)`
добавляет слот в slots коллекцию.
Параметры:
    a. string iden – полностью определенный (full qualified) идентификатор слота. Например, слот age фрейма Elephant будет иметь полностью определенный идентификатор равный Elephant.age;
    b. Slot slot – экземпляр класс Slot;

v. `public static void getSlot (string iden)`
возвращает слот по его полностью определенному идентификатору
Исключения: SlotIsNotFound – данное исключение возникает в случае если слот с заданным идентификатором не найден в коллекции slots

vi. `public static bool isSlot (string iden)`
Проверяет, есть ли слот с заданным идентификатором в коллекции slots или нет.

vii. `public static DataFrame getDataFrame(string iden)`
Возвращает фрейм-экземпляр или фрейм-класс по его идентификатору
Возвращаемое значение: экземпляр InstanceFrame или ClassFrame класса в зависимости от типа фрейма.
Исключения: FrameIsNotFound – данное исключение возникает в случае если фрейм с заданным идентификатором не найден.

viii. `public static RulesetFrame getRulesetFrame(string iden)`
Возвращает «набор правил» по его идентификатору
Исключения: FrameIsNotFound – данное исключение возникает в случае если «набор правил» с заданным идентификатором не найден

ix. `protected static CSharpExpertAbstract getInstance()`
Возвращает экземпляр класса CSharpExpert унаследованного от CSharpExpertAbstract. (Значение атрибута CSharpExpertAbstract.instance)

Методы

i. protected virtual void createFrames()
инициализирует коллекции dataFrames и rulesetFrames. Данный метод доопределяется в классе CSharpExpert добавлением всех фреймов и «наборов правил» определенных в программе.

Abstract Class DataFrame

Это абстрактный класс, который является базовым классом для абстрактных классов InstanceFrame и ClassFrame. В настоящей реализации InstanceFrame и ClassFrame классы никак не расширяют DataFrame класс и введены для будущих расширений. Конкретный фрейм-экземпляр или фрейм-класс наследуется от абстрактного класса InstanceFrame и ClassFrame соответственно.

Атрибуты фрейма копируются в унаследованный экземпляр из кода программы на C# Expert фактически без изменений, меняются только идентификаторы доступа к слотам фреймов используемые в объявлении атрибутов.

Атрибуты

i. `private IList isA`

   список фреймов от которых унаследован данный фрейм. Для экземпляр-фрейма список не может содержать более одного элемента.

Методы

i. `public string getFrameName()`

   Возвращает идентификатор (имя) фрейма.

ii. `public Slot getSlot(string iden)`

   возвращает слот (объект класса Slot) по его идентификатору.

   Параметры: Идентификатор может быть как полностью определенный (например Elephant.age), так и имя слота в области видимости данного фрейма (например age).

   Исключения: SlotIsNotFound – данное исключение возникает в случае если слот с заданным идентификатором не найден.

iii. public void resetSlots()

   Данный метод инициализирует слоты в соответствии с описанием в программе C# Expert. Обычно, данный метод вызывается только при первом запуске экспертной системы.

Class Slot

Данный класс представляет собой реализацию слота фреймовой системы.

Атрибуты

i. `private object current_value`

   представляет собой значение фрейма. При создании слота инициализируется значением null;

ii. `public object slotValue`

   свойство, посредствам которого осуществляется доступ к значению слота

Методы

i. `public virtual void setValue (object new_value)`
   Этот метод переопределяется в слотах, в которых определен параметр restriction. При конвретации кода C# Expert в код C#, содержимое блока restriction просто копируется в тело метода setValue().
ii. `private virtual void ifAdded()`
   вызывается автоматически на событие изменения значения слота с null на не null;
iii. `private virtual void ifRemoved()`
   вызывается автоматически на событие изменения значения слота с не null на null;
iv. `private virtual void ifNeeded()`
   вызывается автоматически при вызове метода getValue()
v. `private virtual void ifModified()`
   вызывается автоматически при вызове метода setValue() если значение меняется не с null на не null, и новое значение отлично от старого

Abstract Class RulesetFrame

Данный класс описывает понятие набора правил языка C# Expert. Конкретный «набор правил» наследуется от абстрактного класса RulsetFrame. Причем атрибуты фрейма копируются из программы на C# Expert в унаследованный экземпляр фактически без изменений, меняются только идентификаторы доступа к слотам фреймов используемые в объявлении атрибутов.

Атрибуты

i. `IDictionary rules`
   Словарь, состоит из правил (слоты типа правило), в качестве ключа используется идентификатор правила, в качестве значения – правило представленное в виде экземпляра класса унаследованного от абстрактного класса Rule.

ii. `IDictionary context`

Задает контекст в котором работает данный набор правил. В качестве ключа используется идентификатор фрейма, в качестве значения ссылка на объект представляющий этот фрейм

Abstract Class Rule

Данный класс представляет собой правило из набора правил. Конкретное правило это унаследованный класс on абстрактного класса Rule у которого переопределены абстрактные методы if_statement(), else_statement() и condition()

Методы

1. `public bool condition()`
   возвращает значение истинно ли условие правила или нет
2. `public void if_statement()`
   вызывает этот метод в случае если условие истинно
3. `public void else_statement()`
   вызывает этот метод в случае если условие ложно

Машина логического вывода

Машина логического вывода реализована в виде независимого класса ProductionSystem. Данный класс предоставляет следующий свойства и методы:

i. public static bool showComments – данное свойство задает, будут ли показываться комментарии определенные в правилах в ходе консультации или нет;
ii. public static void consult (string ruleset);
   public static void consult (string ruleset, string goal);
   public static void consult (string ruleset, string goal, ChainingMehod method);
   Данный метод начинает консультацию используя заданный набор правил.
   Параметры

ruleset – имя набора правил, используя который будет осуществляться консультация;
goal – имя цели, может быть либо имя слота из контекста набора правил, либо имя параметра. Данный параметр не обязателен, если цель не определен, выводиться цель, определенная в наборе правил;
method – стратегия логического вывода, может быть прямой или обратной (По умолчанию используется прямой вывод).

Особенности текущей реализации машины вывода:

i. В настоящее время, машина вывода поддерживает только прямую стратегию вывода;
ii. Метод разрешения конфликтов используемый машиной заключается в выборе правила, имя которого будет первым исходя из лексикографического порядка;

Нечеткие типы данных

В языке C# Expert предусмотрена возможность представления нечетких знаний. В качестве способа выражения нечеткости выбраны коэффициенты уверенности как наиболее простой и понятный способ. Поддержка нечетких типов осуществляется на уровне библиотеки. Для представления нечетких знаний определен базовый класс FuzzyType. В классе FuzzyType объявлено свойство public int FuzzyType.Cf – коэффеициент уверенности. Коэффициент уверенности может принимать значения лот 0 до 100.

От класса FuzzyType унаследованы следующие классы:

i. FuzzyBool;
ii. FuzzyDouble;
iii. FuzzyInt;
iv. FuzzyString;

Над нечеткими типами определены такие же операции, как и над обычными значениями. Однако эти операции, кроме собственно вычисления результата, вычисляют также его коэффициент уверенности по следующим правилам:

i. Для арифметических операций, операций отношения и логического «и» коэффициент уверенности полагается равным минимальному из коэффициентов операндов;
ii. Для логического отрицания коэффициент уверенности полагается равным единице минус коэффициент уверенности операнда;
iii. Для логического «или» коэффициент уверенности вычисляется как максимальное значение из коэффициентов уверенности операндов;

Требуемая спецификацией функциональность реализована с помощью перегрузки операций. Также используется неявное преобразование обычных и нечетких типов между собой, что позволяет удобно использовать их вместе.

В случае необходимости, инженер знаний может создать свой нечеткий тип как подкласс FuzzyType, перегрузив необходимые операции.

Заключение

В ходе выполнения диплома, проделаны следующие работы:

v. Разработан синтаксис и семантика языка C# Expert;
vi. Описана грамматика языка с помощью пакета для создания компиляторов CoCo/R;
vii. Реализован рабочий прототип метакомпилятора C# Expert с поддержкой основной функциональности языка;

Реализовано несколько несложных примеров на C# Expert, которые демонстрируют общие принципы использования языка;

Примеры

В качестве демонстрации возможностей языка C# Expert было создано три простых примера экспертных систем. Исходные коды примеров прилагаются вместе с исходными кодами метакомпилятора C# Expert и библиотеки. Данный раздел содержит краткое описание примеров.

Пример 1. Определение качества стекла (GlassExpert)

В данном примере рассматривается экспертная система, перенесенная из GURU по определению качества стекла. Пример содержит более 20 правил, которые хорошо демонстрируют возможности машины логического вывода языка представления знаний C# Expert. Также в примере создано несколько фрейм-классов (Стекло, Шихта, Песок), которые, в свою очередь, показывают возможности C# Expert по структурированию данных. В исходной экспертной системе все значения хранились в обычных переменных в глобальном пространстве. Также существенный недостаток исходной системы это использования целых чисел для индикации состояния системы. В варианте перенесенном на C# Expert использование целых чисел было заменено на использование перечислений (enum) что значительно увеличило читабельность исходного кода.

Пример 2. Пример интеграции GUI с экспертной системой

Пример “Sample Of Expert System 1 (Math)” – это очень простой пример, содержащий всего 4 правила, основное назначение его продемонстрировать легкость интеграции БЗ с графическим интерфейсом. Также в данном примере показывается возможность сохранения БЗ между сеансами.

Пример 3. Пример использования делегата в качестве типа слота

Пример “CottagePriceCalculator” показывает возможность использования делегата в качестве типа слота. Делегаты позволяют гибко использовать процедурные знания в БЗ экспертной системы.

Приложения

Приложение 1. Грамматика языка C# Expert

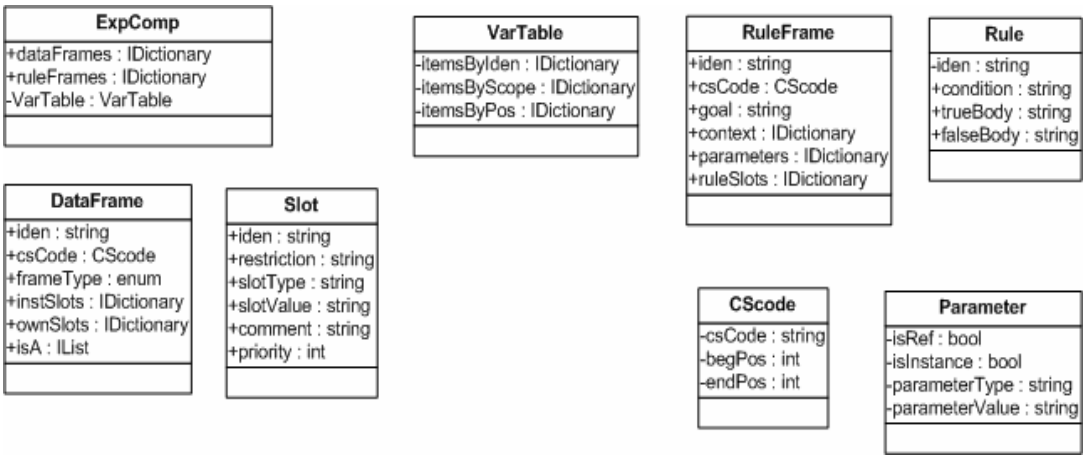
Грамматика C# Expert является расширением грамматики C# version 1.2. Нижеприведенный листинг содержит определения только определение лексем языка C# Expert. Полное описание грамматики смотрите в файле “CSharpExpert.ATG”.

```
      CS
      =
        {UsingDirective}
        {IF (IsGlobalAttrTarget()) GlobalAttributes}
        {NamespaceMember}
        ["#frames" {ExpertFrameDeclaration}].

ExpertFrameDeclaration = "frame" ( ("class" ident ExpertFrameClassBody)|
      ("instance" ident ExpertFrameInstanceBody) |
      ("ruleset" ident  [ExpertRulesetParamList] ExpertFrameRulesetBody)
      ).


ExpertFrameClassBody = "{" [ExpertFrameAttributes<FrameType.dataFrame>]
[ExpertOwnClassSlots] [ExpertInstanceSlots] "}".

ExpertFrameInstanceBody = "{"
[ExpertFrameAttributes<FrameType.dataFrame>] ExpertOwnInsSlots "}".

ExpertFrameAttributes<FrameType frTp> ={{Attributes} {MemberModifier<m>}
        ClassMember<m>}.

ExpertOwnClassSlots =
      "own_slots" [ExpertOwnClassIsA] {ExpertSimpleOwnSlot |
ExpertFacetOwnSlot}.

ExpertOwnInsSlots =
      "own_slots" ExpertOwnInsIsA {ExpertSimpleOwnSlot |
ExpertFacetOwnSlot}.

ExpertInstanceSlots =
      "instance_slots" {ExpertSimpleInsSlot | ExpertFacetInsSlot}.

ExpertOwnInsIsA =
      is_a" ident ";".

ExpertOwnClassIsA =
      "is_a" ident [ "," ident] ";".

ExpertType<out string val> =
      (SimpleType | Qualident<out val> | "object" | "string").
```

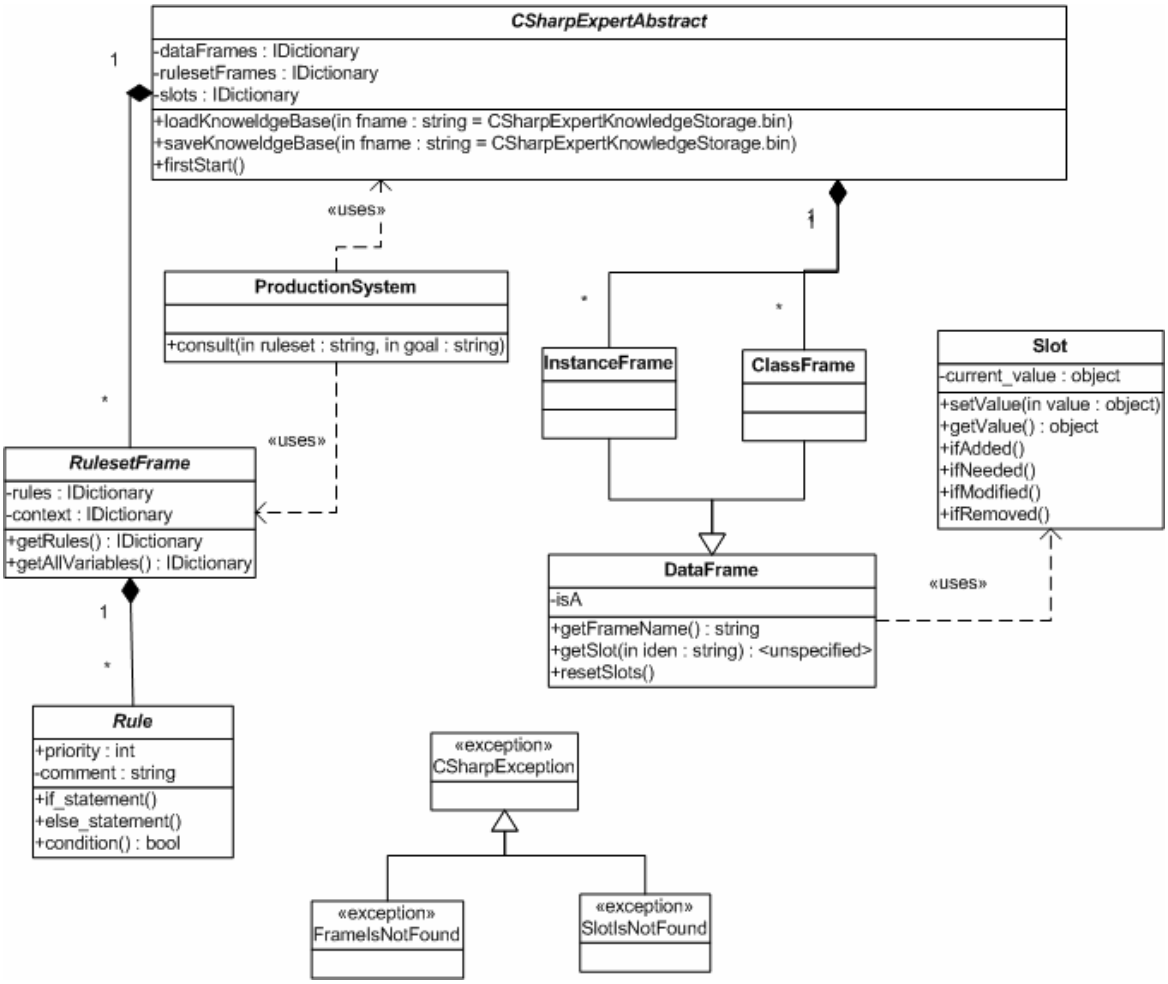
```
ExpertSimpleOwnSlot =
            (
            (IF(!IsAssignment())
            ExpertType<out slotType> ident ["=" Expr])  | (ident "="
            Expr)
      ) ";".

ExpertFacetOwnSlot =
            "facet" "{" "type" ExpertType<out slotType> ";" "value"
Expr";"
            ["restriction" Expr ";" ]
      "}"
            ident.


ExpertSimpleInsSlot =
                  ExpertType<out slotType> ident["="Expr]";".

ExpertFacetInsSlot =
            "facet" "{"
            "type"
            ExpertType<out slotType>
                  ";"
            ["default_value" Expr";"]
            ["restriction" Expr";"]
      "}" ident";".

ExpertRulesetParamList =
      "(" ExpertRulesetParam {"," ExpertRulesetParam} ")" .

ExpertRulesetParam =
            ["ref"] ExpertType<out parameterType> ident.

ExpertFrameRulesetBody =
      "{"
            [ExpertFrameAttributes<FrameType.rulesetFrame>]
            "own_slots"
            ExpertRulesetSlots.

ExpertRulesetContextList =
      "context" ExpertRulesetContextParam { ","
ExpertRulesetContextParam}";".

ExpertRulesetContextParam = ["instance"] ident.

ExpertRulesetGoal = "goal" ident";".

ExpertRulesetSlots = ExpertRulesetContextList ExpertRulesetGoal
{ExpertRule}.

ExpertRule =
      "rule" "{"
      ExpertIfThenElse<rule>
      [ "comment" stringCon]
      ["priority" intCon]
      "}" ident ";".

ExpertIfThenElse<Rule rule> =
      "if" "(" Expr ")"
```

```
"then" EmbeddedStatement
["else" EmbeddedStatement].
```

Приложение 2. UML диаграмма промежуточного кода



Приложение 3. UML диаграмма библиотеки CSharpExpertLibrary



Приложение 3. Исходные коды

К данной дипломной работе прилагаются исходные коды системы C# Expert и примеры. (Архивный файл: “CSharpExpert20040601.zip”). Краткое содержание архива:

i. bin\ – откомпилированные исходные коды;
ii. src\ – исходные коды библиотеки и компилятора
iii. examples\ – три примера экспертный систем, написанных на C# Expert;

Литература

[1] Сафонов В.О, докторская диссертация;

[2] Л. Растригина, журнал "Радио" №6-1988;

[3] Альфред Ахо, Рави Сети, Джеффри Ульман «Компиляторы. Принципы, технологии, инструменты». Изд-во Вильямс, 2001;

[4] Том Арчер “Язык C#”;

[5] «Системы искусственного интеллекта» (http://penguin.photon.ru/doc/ai.shtml);

[6] Allison Cawsey, ``Developing an Explanation Component for a Knowledge-Based System: Discussion", In Expert Systems with Applications, vol 8:4 pp 527-531, 1995;

[7] Allison Cawsey, “Databases and Artificial Intelligence 3” , Chapter Knowledge Representation and Interfaces (http://www.cee.hw.ac.uk/~alison/ai3notes/all.html);

[8] POPLOG-FLEX reference (http://www.brunel.ac.uk/research/AI/ajay/project/flex-help/main-help-contents.html);

[9] Amzi! Inc., “Building Expert Systems in Prolog” (http://www.amzi.com/ExpertSystemsInProlog/);

[10] H. Mossenbock, ETH Zurich, “CocoR / A Generator for Fast Compiler Front-Ends. Technical Report”;

[11] Hassan Gomaa, Larry Kerschberg “Domain Modeling for Software Reuse and Evolution”