УДК 681.3

А.Н. Петрушенко, В.А.Хохлов, А.О. Плотников, И.А.Петрушенко

МОДУЛЬНАЯ АРХИТЕКТУРА ДИАЛОГОВОЙ ТРАНСФОРМАЦИОННОЙ МАШИНЫ «ДИАМАНТ» И СИСТЕМА ДОСТУПА К БАЗЕ ПРЕДМЕТНОЙ ОБЛАСТИ

Введение. Работа выполнена в рамках исследований по созданию трансформационных методов синтеза знаний в произвольных предметных областях - перспективного направления информатики, которое развивается как в Украине, так и за рубежом. Согласно с трансформационными методами программы получаются путем поэтапного преобразования входного описания системы по правилам, совокупность которых представляет собой знания о задаче, которая решается. В данной области исследований можно условно выделить два взаимосвязанных направления - теоретическое и прикладное. Теоретическое направление требует, в частности, предъявления различных моделей вычислений, охватывающих те или иные конструкции алгоритмических языков, и изучение базовых трансформаций в этих моделях. Так, наличие полной системы независимых трансформаций свидетельствует о полной изученности класса алгоритмов, хорошими системами являются системы, обладающие свойством Чёрча-Россера и так далее. Прикладной направление связывают с созданием трансформационной машины, командами для которой служат базовые трансформации, а данным - выражения в том языке, над которым эти трансформации заданы.

Украинская, прежде всего киевская, алгебро-кибернетическая школа проводит исследования модели вычислений, в основе которой лежит понятие абстрактно-автоматной модели кибернетической системы (АМКС), функционирующий по принципу взаимодействия с обратной связью двух автоматов: управляющего и операционного или, как еще его называют, информационного. Представление вычислительного процесса в виде АМКС позволило В.М.Глушкову и его ученикам создать новое теоретическое направление в прикладной теории алгоритмов - алгебру алгоритмов, позволяет формализовать общую идею трансформационного синтеза [1, 2]. Однако большая системная интегрированность и, как следствие, абстрактность получаемых результатов входит в противоречие с недостаточной их содержательностью (интенсиональностью) в процессе решения конкретных задач. На удовлетворение потребности в интенсиональной теории вычислительных процессов направлены исследования в рамках композиционного программирования, программологии и дескриптологии, целью которых является адекватное уточнение сущности программирования, в частности, через раскрытие сущности отношений именования, которые, как постулируется, составляют характерную черту программирования и определяют его нетрадиционную сущность [3-6].

Постановка задачи. ДТМ является инструментарием синтеза знаний в различных предметных областях на основании алгебры алгоритмов, композиционного программирования, программологии и дескриптологии [7, 8]. Основными компонентами ДТМ являются базы знаний и машины вывода. В данной работе дальнейшее развитие получает архитектура базы знаний ДТМ и подсистемы ДТМ, ориентированные на хранение и обработку предметной области и объектов, связанных с нею.

Модульный подход к организации архитектуры ядра современных программных продуктов в настоящее время пользуется все большей популярностью за счет неоспоримых преимуществ по сравнению с традиционным подходом [9-11]. Ядро системы, построенной на модульной архитектуре, обеспечивает механизм загрузки модулей, которые поддерживают различные функции и библиотеки. Существуют два способа организации загрузки модулей: статический и динамический. Динамическая загрузка модулей выполняется «на лету», без перезагрузки программы, непосредственно в работающей системе. Статическая загрузка, в свою очередь, предполагает реконфигурацию системы и загрузку модулей после ее рестарта.

Модульное ядро имеет свои преимущества, как для разработчика, так и для пользователя разрабатываемых систем. Разработчики могут не выполнять полную перекомпиляцию ядра системы при подключении к нему новых модулей. Кроме того, тестирование, отладка, локализация системы с подключенными модулями значительно упрощается. Однако главным преимуществом использования модульной архитектуры в программных приложениях является легкость расширения функциональности системы, в том числе и для сторонних разработчиков.

К недостаткам модульного подхода реализации архитектуры программного обеспечения относится, в первую очередь, сложность разработки и проектирования. Такая архитектура требует чрезвычайно хорошо продуманной основы ядра, механизмов наращивания, совместимости.

Решение задачи. При проектировании архитектуры ДТМ «Диамант» был выбран модульный подход, благодаря чему ядро системы представляет собой менеджер модулей, который осуществляет подгрузку/выгрузку нужных модулей. Менеджер модулей описан в классе PluginManager. Данный класс начинает

свою работу при запуске системы и выполняет загрузку библиотек из двух директорий: директории модулей и компонентов. Менеджер имеет функции загрузки/выгрузки модулей, сортировки их по типу для оптимизации поиска.

Загрузка модулей выполняется по следующему алгоритму: задается путь к библиотеке модуля, в заданной директории выполняется поиск контрольного символа GetPluginInstance – функции получения экземпляра класса модулей. Успешный поиск показывает, что библиотека является модулем ДТМ. В этом случае выполняется получение экземпляра модуля и сохранение его, а также информации о нем в специальной структуре, которая является основным хранилищем данных класса PluginManager. Согласно этой информации менеджер предоставляет возможность получения некоторого количества модулей с указанием их типа, а также отдельно взятого модуля.

На верхнем уровне иерархии интерфейсов модулей системы находится абстрактный класс iPlugin. Он содержит единственную функцию GetPluginInfo - функцию получения информации о модуле. Информация представляется в виде структуры PluginInfo, содержащей следующие данные:

- имя модуля;
- версия;
- описание модуля;
- тип модуля;

Все модули, подгружаемые в систему, должны быть порождены от интерфейса iPlugin – прямое наследование невозможно, так как потребуется введение нового типа модулей. Менеджер модулей работает только с классом iPlugin, обращаясь к функциям классов-потомков с помощью механизма виртуальных функций C++.

В данной статье рассматривается одна из реализаций модульной системы в ДТМ «Диамант» - система доступа к базе предметной области.

Базовым для модулей доступа к предметной области является абстрактный интерфейс IKBaseAccessor, не имеющий реализации. Потомком интерфейса IKBaseAccessor является IKBaseStructureAccessor, который конкретизирует IKBaseAccessor операциями над объектами базы данных (рисунок 1).

Рис.1. Дерево интерфейсов доступа к предметным областям системы ДТМ «Диамант»

Каждый модуль доступа к предметной области, базирующийся на операциях с конкретным хранилищем данных, должен быть порожден от интерфейса IKBaseAccessor (в этом случае программисту дается возможность самостоятельно проектировать структуру базы данных, определять объекты базы данных и связи между ними) или от интерфейса IKBaseStructureAccessor (в этом случае программисту необходимо только реализовать операции над объектами базы данных, используя инструменты конкретной СУБД).

Методы, предоставляемые интерфейсом IKBaseAccessor:

- open/close – открытие/закрытие базы данных, хранящей предметную область;
- create – создание базы предметной области. От реализации данного метода ожидается создание базы данных определенной структуры;
- sdoGroups – получение списка групп предметных областей (в виде пар имя-идентификатор);
- sdoNames – получение списка предметных областей по заданной группе предметных областей;
- opIds – получение списка идентификаторов операторов для заданной предметной области, заданного типа данных и соответствующих заданному атрибуту доступа (открытый/закрытый/защищенный);
- elTypeNames – получение списка элементарных типов данных для заданной предметной области;

- elStructuredTypeNames – получение списка составных типов данных для заданной предметной области;
- opRealisation – получение реализации заданного оператора для указанного типа из указанной предметной области;
- typeRealisation – получение реализации заданного типа из указанной предметной области;
- usedSDOs – получение списка предметных областей, используемых заданной предметной областью;
- handleSDO – получение идентификатора предметной области по ее имени;
- handleType – получение идентификатора типа данных по его имени;
- opVars – получение списка переменных, используемых оператором;

Методы, ориентированные на операции с базой знаний:

- create – создание новой базы знаний;
- addSdoGroup – подключение к редактору базы знаний заданной по имени групп предметных областей;
- addSdo – подключение к редактору базы знаний заданной по имени предметной области;
- addElementaryType – создание нового элементарного типа в заданной базе знаний (указываются именительный и родительный падеж имени нового типа);
- addDataMember – добавление нового элемента в составной тип данных - указываются идентификаторы составного типа данных, его предметной области, добавляемого элемента и его предметной области, имя нового элемента и спецификатор доступа к элементу (закрытый, защищенный, публичный);
- addOperator – добавление нового оператора для заданного типа – указываются идентификаторы типа и его предметной области, тип возвращаемого оператором значения и спецификатор доступа к элементу;
- setOpRealisation – установка реализации данного оператора – указываются идентификаторы оператора, его предметной области, язык реализация оператора и реализация оператора в виде строки;
- addUsedSDO- добавление предметной области в список предметных областей, используемых другой предметной областью;
- addParent – добавление родителя указанному типу. Кроме стандартных идентификаторов здесь указывается спецификатор доступа к элементам родительского типа – закрытое наследование, открытое наследование, защищенное наследование.

В качестве примера рассмотрим реализацию некоторых функций данного интерфейса применительно к СУБД SQLite.

На рисунке 2 представлена структура базы данных, предназначенной для хранения предметной области и связанной с ней объектов и реализованной с использованием СУБД SQLite.

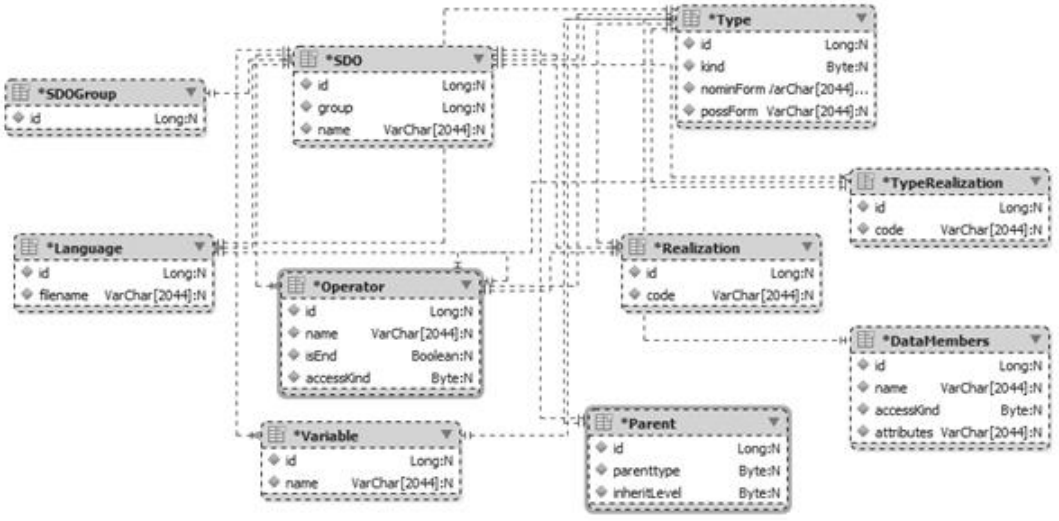

Рис. 2. Структура БД для хранения предметной области, и объектов, связанных с нею в ДТМ «Диамант», реализованная в СУБД SQLite.

Назначение представленных на рисунке 2 таблиц и логический смысл хранимых в них данных:

- SDO – предметная область;
- SDOGroup – группа предметных областей;
- Type – элементарный тип данных;
- Operator – элементарный оператор;
- Realization – реализация оператора;
- TypeRealization – реализация типа данных;
- DataMembers – элементарные элементы сложного типа данных;
- Parent – родитель элементарного типа данных, используется для реализации наследования;
- Variable – переменная оператора.
- Language – язык реализации оператора.

Как пример обработки хранимых данных, используя СУБД SQLite, рассмотрим функции для создания нового элементарного типа данных и добавления родителя указанному типу данных.

Создание нового элементарного типа данных:

```
HTYPE KBaseASqlite::addType(HDOMAIN iHSdo, TYPEKIND iKind,
      const std::wstring &iNominForm, const std::wstring &iPossForm)
{
   if (!pDb_)
      throw (KBException(KBException::EID_NOTOPEN));

   std::wostringstream sqlst;
   sqlst << L»INSERT INTO types VALUES(NULL, « << iHSdo <<  ‘,’ <<
      iKind << «,\’» << iNominForm << «\’,\’» << iPossForm << «\’)»;
   if (sqlite3_exec(pDb_, wstr2utf8(sqlst.str()).c_str(), 0, 0, 0) != SQLITE_OK)
      throw KBExceptionSQLite(KBException::EID_ADDTYPE, sqlite3_errmsg(pDb_));

   return static_cast<HTYPE>(sqlite3_last_insert_rowid(pDb_));
}
```

Добавление родителя указанному типу данных:

```
Void KBaseASqlite::addParent(HDOMAIN iHSDO, HTYPE iHType, IHDOMAIN iIHSDO,
          HTYPE iHParentType, INHERLEVEL iInherLevel)
{
   if (!pDb_)
      throw (KBException(KBException::EID_NOTOPEN));

  std::wostringstream sqlst;
   sqlst << L»INSERT INTO parents VALUES(« << iHSDO << ‘,’ <<
      iHType <<  ‘,’ << iIHSDO << ‘,’ << iHParentType << ‘,’ <<
      iInherLevel << ‘)’;
   if (sqlite3_exec(pDb_, wstr2utf8(sqlst.str()).c_str(), 0, 0, 0)
         != SQLITE_OK)
      throw KBExceptionSQLite(KBException::EID_ADDPARENT, sqlite3_errmsg(pDb_));
}
```

Выводы. По результатам проведенных исследований установлено, что для решения поставленной задачи наиболее приемлемым является использования модулей, которые подгружаются в ядро системы, представляющее собой менеджер модулей. Использование модульной архитектуры ДТМ «Диамант» позволяет существенно увеличить гибкость данной системы и использовать наиболее эффективную конфигурацию для решения конкретных задач. Также данный подход позволяет абстрагировать ядро ДТМ от специфики хранения данных в конкретной СУБД, обращаясь в процессе работы к функциям интерфейса IKBaseAccessor. Используя интерфейс IKBaseStructureAccessor, располагающий предопределенной структурой базы знаний, программист освобождается от необходимости ее проектирования, и может сосредоточиться лишь на программировании набора низкоуровневых операций над объектами базы данных.

ЛИТЕРАТУРА.

1. Глушков В.М., Цейтлин Г.Е., Ющенко Е.Л. Алгебра. Языки. Программирование. 3-е изд., перераб. и доп. - К.: Наукова думка, 1989. - 376 с.
2. Капитонова Ю.В., Летичевский А.А. Математическая теория проектирования вычислительных систем. - М.: Наука, 1988. - 296 с.
3. Басараб И.А., Никитченко Н.С., Редько В.Н. Композиционные базы данных. – Киев: Либідь, 1992. – 191 с.
4. Редько В.Н. Программология: прошлое, настоящее, будущее // Вестник Международного Соломонова университета. - 1999. - № 1. - С.23-59.
5. Редько В.Н. Основания дескриптологии //Кибернетика и системный анализ. – 2003. - № 5. – С. 16 – 36.
6. Петрушенко А.Н. Очерки по методологии научного познания: от математических к информационным моделям мира. – К.: Наукова думка, 1998. - 119 с.
7. Петрушенко А.Н., Хохлов В.А., Ткачев В.А., Шепетухин Е.С. Диалоговая трансформационная машина: некоторые функциональные возможности // Проблемы программирования. – 2000. – № 1–2 (Спец. выпуск) – С. 323–334.
8. Петрушенко А.М., Хохлов В.А. Концепція діалогових обчислень та деякі проблеми автоматизації програмування // Там же. – 2004. – № 2–3 (Спец. выпуск) – С. 37–47.
9. Г. Буч. Объектно-ориентированный анализ и проектирование с примерами приложений на C++, 2-е изд./ Пер. с англ. – М.: «Издательство Бином», Спб.: «Невский диалект», 1999. – 560 с.
10. UML. Проектирование систем реального времени, параллельных и распределенных приложений. Пер. с англ. – М.: «ДМК Пресс», 2002. – 704 с.
11. Зиглер К. Методы проектирования программных систем. – М.: Мир, 1985 – 328с.

ПЕТРУШЕНКО Анатолий Николаевич, к.ф-м.н., доцент кафедры информационных технологий ХНТУ.

Научные интересы: синтез знаний в сложных предметных областях на основании концепции диалоговых вычислений в формальных системах и аппарата алгебры алгоритмов и программологии.

ХОХЛОВ Вадим Анатольевич, к.т.н..

Научные интересы: инструментальные средства программирования, автоматизация процесса разработки программного обеспечения, параллельные вычисления.

ПЛОТНИКОВ Алексей Олегович, ассистент кафедры информационных технологий, ХНТУ.

Научные интересы: автоматизация проектирования аппаратного обеспечения компьютеров, прикладное программирование.

ПЕТРУШЕНКО Иван Анатольевич, студент четвертого курса факультета кибернетики КНУ им. Т.Г.Шевченко

Научные интересы: теория и технология программирования