Уважаемые члены государственной аттестационной комиссии вашему вниманию предлагается дипломный проект на тему «Разработка процедурного языка программирования в системе имитационного моделирования РДО» выполненный студентом группы РК9-Д2 Чирковым Михаилом Михайловичем, руководитель дипломного проекта: Урусов Андрей Витальевич.

Сначала несколько слов об имитационном моделировании, в общем, и системе имитационного моделирования РДО в частности.

Имитационное моделирование предназначено для прогноза поведения сложных систем, а так же результатов их взаимодействия. Дискретно – событийное моделирование подразумевает такой подход к моделированию, при котором абстрагируются от непрерывности происходящих событий. При дискретно – событийном моделировании считается, что состояние системы меняется мгновенно через сколь угодно короткие промежутки времени. Но в промежутке между двумя ближайшими событиями состояние системы остаётся неизменным.

Язык имитационного моделирования РДО поддерживает дискретно-событийную парадигму имитационного моделирования и реализует три основных подхода моделирования: процессно-ориентированный подход, событийный подход, подход сканирования активностей.

Про реальные системы можно сказать, что они локально императивны. То есть, если взять достаточно узкую задачу, то ее можно вполне легко описать методами процедурного программирования. Процедурное программирование наиболее пригодно для реализации небольших подзадач, где очень важна скорость исполнения. Особенность процедурного программирования состоит в том, что задачи разбиваются на шаги и решаются шаг за шагом.

В РДО отсутствовала возможность использования основных операторов процедурного программирования.

На листе постановки задачи показано, что данная разработка упрощает создание имитационных моделей и существенно сокращает объем кода модели.

На этапе концептуального проектирования был изучен процесс компиляции исходных файлов RAO-Studio, представленный на IDEF0 диаграммах. Входными данными являются: исходный код на языке C++, файл грамматики языка РДО и лексемы языка РДО. Выходными данными является исполняемый файл RAO-Studio. Файлы грамматики языка РДО и лексемы языка РДО, с помощью лексического анализатора «bison», преобразуются в исходный код на я зыке С++. Исходный код языка С++ собирается в статические библиотеки с расширением \*.lib, которые, в свою очередь, собираются в исполняемый файл RAO-Studio. Так же была изучена внутренняя структура системы имитационного моделирования РДО, которая отображена на листе «диаграмма компонентов».

rdo\_parser производит лексический и синтаксический разбор исходных текстов модели, написанной на языке РДО. При разработке были добавлены компоненты, реализующие поддержку нового синтаксиса описания моделей.

rdo\_runtime отвечает за непосредственное выполнение модели, управление базой данных и базой знаний. В процессе разработки были добавлены компоненты, реализующие логику работы операторов процедурного программирования и описывающие экземпляры нового типа данных в языке РДО.

На этапе технического проектирования был разработан синтаксис описания нового типа данных, а так же синтаксис описания операторов процедурного программирования.

Описания типа данных, представлено на синтаксической диаграмме. В РДО существую следующие типы данных: целочисленный, вещественный, перечислимый, строковый, логический, ссылка на один из ранее описанных типов данных и, вновь разработанный, тип данных массив. Массив начинается с ключевого слова «array», после которого в треугольных скобках указывается тип данных элементов массива. Поскольку типом данных элемента может являться массив, это позволяет создавать массивы неограниченной вложенности.

В данном дипломном проекте были разработаны операторы процедурного программирования, синтаксис описания которых представлен на синтаксической диаграмме алгоритмической функции.

Описание циклического оператора начинается с ключевого слова «for», за которым в круглых скобках указывается оператор присваивания начального значения циклической переменной или её объявление, как локальной переменной. Затем следует логическое условие выхода из цикла, потом указывается арифметическое выражение приращения циклической переменной. В теле цикла указывается исполняемый оператор.

С помощью оператора присваивания переменным задаются новые значения. Описание оператора начинается с имени переменной, затем следует простой или составной оператор присваивания, после которого указывается арифметическое выражение, которое может являться константой. Описание оператора должно заканчиваться точкой с запятой.

Описание оператора выбора начинается с ключевого слова «if», затем следует логическое выражение. Если выражение истинно, выполняется оператор, указный после скобок. Если ложно, выполняется оператор, указанный после ключевого слова «else».

Добавлена поддержка локальных переменных, область локальной памяти выделяется фигурными скобками.

Описание оператора возврата начинается с ключевого слова «return», после которого указывается возвращаемое значение.

Оператор прерывания описывается ключевым словом «break» и реализует выход из текущего цикла.

На диаграммах активностей представлено описание алгоритма создания экземпляра типа массив, алгоритма определения наличия оператора возврата и алгоритма, выполнения оператора алгоритмической функции.

На этапе рабочего проектирования были внесены необходимые изменения в файлы синтаксического анализатора.

В пространстве имен rdoRuntime были добавлены классы, необходимые для описания нового типа данных массив и реализации функционирования операторов процедурного программирования. Место новых классов во внутренней структуре РДО представлено на диаграммах классов.

В пространстве имен rdoParser были добавлены классы необходимые для лексического анализа описания нового типа данных массив.

В исследовательской части данного дипломного проекта был произведён анализ различных вариантов реализации обработки оператора возврата и оператора прерывания.

В рамках данной задачи было рассмотрено два варианта обработки операторов прерывания и возврата.

Первый вариант реализации подразумевает использования обработчика исключений языка С++. Второй вариант реализации при своей работе вместо исключений использует дополнительные проверки. Логика реализации данных вариантов представлена на диаграммах активности.

Для каждого варианта реализации была написана тестовая программа, текст программ приведён в приложении. Было произведено по 10 измерений для разного количества итераций, от 10 тысяч до 300 тысяч с шагом в 10тысяч итераций. По результатам измерений быстродействия был выбран второй, основанный на дополнительных проверках. Он показал результаты на два порядка лучше, что можно наблюдать на графиках зависимости времени исполнения от количества итераций.

На листе экранных форм представлены скриншоты описания тестовой модели, а так же скриншот прогона модели, что подтверждает работоспособность программного продукта после внедрения разработки.

На листе результатов представлен участок описания функций ранее разработанной имитационной модели почтового отделения до внедрения разработки и после. Ниже представлены результаты моделирования до и после внедрения. Из их идентичности можно сделать вывод об адекватности внесенных изменений, т.е. данная разработка не нарушила логику функционирования RAO-Studio.

В заключении можно заметить, что данная разработка позволила упростить реализацию сложных алгоритмических зависимостей при описании имитационных моделей, существенно сократив исходный код модели.