СИСТЕМА ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ ДИНАМИКИ ПРОЦЕССОВ В МЕЖКУЛЬТУРНОЙ КОММУНИКАЦИИ

А.А. Шутов,
старший преподаватель, Нижегородский филиал Государственного университета-Высшей школы экономики
А.А. Долгов,
ООО «Теком», инженер-исследователь
О.Р. Козырев,
к.ф-м.н., профессор Государственного университета-Высшей школы экономики

Описаны архитектуры и методы реализации распределенной программной системы имитационного моделирования динамики процессов в межкультурной коммуникации. В проекте разработаны теоретические основы математико-информационного моделирования динамических процессов и описана реализация многоагентной распределенной имитационной модели.

Введение

Исследователями в области социальной динамики отмечается, что мировое сообщество переживает системный кризис, для которого характерны неравновесная динамика большинства систем и большое влияние бифуркаций. В таких условиях важнейшую роль при анализе социально-экономической ситуации, выборе оптимальной стратегии развития играет всестороннее математическое моделирование с учётом влияющих факторов и активное использование вычислительных комплексов для проведения имитационных экспериментов с альтернативными сценариями.

Это приводит к необходимости организации программной разработки систем имитационного моделирования. Построенные математические модели должны обладать свойством переносимости с одной системы моделирования на другую, уметь объединяться с независимо созданными моделями из других предметных областей. Особые требования предъявляются к моделям социальных и экономических явлений. Они должны быстро интегрироваться с прикладными системами поддержки решений, планирования и управления на уровне предприятия, региона или целой отрасли.

Для эффективного разрешения указанных выше проблем требуются информационно-вычислительные средства нового поколения на основе современных технологий компонентных и распределенных вычислений. Они позволяют исследователям в области социально-экономической теории и практики проводить коллективную разработку и интеграцию математических моделей вне зависимости от особенностей программной реализации, с последующей автоматизированной генерацией кода для проведения численных экспериментов в различных системах имитационного моделирования.

Нами в рамках различных проектов ведутся работы по созданию информационно-вычислительных систем имитационного моделирования. Практическая реализация такой системы – результат общих исследований. Разработка единой системной методологии необходима всем этапам математического моделирования, проектирования и реализации программного обеспечения.

Описание проблемы

Цель исследования проекта «Система имитационного моделирования динамики процессов в межкультурной коммуникации» – разработка теоретических основ математико-информационного моделирования динамических процессов

в межкультурной коммуникации в форме многоагентной распределенной имитационной модели и сопутствующего набора онтологий.

Задачи:

- ✧ построение функциональной, а затем и математической модели;
- ✧ построение информационной (имитационной) модели, связывающей разработанные математические модели с подтверждающими фактическими данными.

Оригинальной формой реализации информационной модели является информационно-логическая система, состоящая из:

- ✧ подсистемы построения и хранения декларативных описаний семантики математических моделей межкультурной коммуникации (банк знаний по схемам);
- ✧ подсистемы генерации кода и сопровождения жизненного цикла программного обеспечения для моделирования (автоматизированная среда разработки);
- ✧ подсистемы настройки и управления проведением имитационных экспериментов (имитационный сервер), объединяющей интерфейсную и справочную подсистемы (Web-портал).

Постановка задачи

1. Построить математическую модель и алгоритм семантического анализа онтологий.

2. Разработать распределённую архитектуру взаимодействия иммитационного сервера и клиентов.

3. Разработать средства для автоматического создания иммитационных моделей.

Описание реализации

Основное содержание проведённой работы:

1. Разработаны проект серверной части CORBA-интерфейса к имитационному серверу на основе системы SWARM, проектная документация и набор прототипов клиентских приложений с CORBA-интерфейсом к имитационному серверу.

2. Построена математическая модель и алгоритм семантического анализа онтологий; выбраны программные технологии для практической реализации. Результаты исследований представлены на международной конференции «Кограф 2006», (Нижний Новгород, ноябрь 2006 г.) и опубликованы в трудах конференции.

3. Проведён сбор и предварительная подготовка материалов для методического пособия по созданию имитационных моделей в системе SWARM.

Разработка распределённой архитектуры взаимодействия иммитационного сервера и клиентов

Данная задача – продолжение исследований в области применения многоагентных методов моделирования на базе библиотеки SWARM. В предыдущих работах проанализирован выбор программных продуктов для реализации клиентской и серверной частей иммитационной системы.

Архитектура распределённой системы объединяет в себе: сервер, реализованный на Swarm, и клиент, реализованный на Repast. Поэтому для начала необходимо выбрать способ взаимодействия этих составляющих в сети. В качестве такого способа выбор остановился на архитектуре CORBA.

CORBA (Common Object Request Broker Architecture) – стандарт, применяющийся с 1991 г. Преимущество его в том, что он определяет, как приложения будут взаимодействовать независимо от того, где они запущены и на каком языке программирования они написаны. В действительности, программист вообще не заботится о сетевом взаимодействии – необходимо лишь реализовать одинаковые интерфейсы на взаимодействующих машинах – в нашем случае клиента и сервера (*рис. 1*).

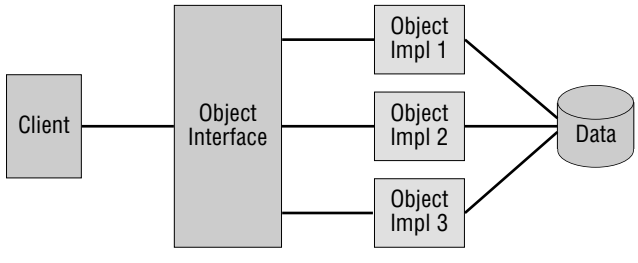

Рис. 1. Общая программная архитектура распределенной системы

Этот рисунок отображает стандартное трехуровневое устройство архитектуры клиент-сервер. Данная архитектура состоит из трёх типов объектов: клиент, объект и источник данных. Источник данных выступает в роли сервера, причём нам неважно, как устроен внутри этот объект.

Клиент и сервер в нашем случае написаны на разных языках программирования. В случае использования CORBA определение интерфейса происходит независимо от его реализации. Поэтому один интерфейс может быть реализован на разных языках, и в каждой реализации объекты будут обращаться не к объектам, а именно к базовому интерфейсу (*см. рис .2*)

Для написания интерфейсов используется специальный язык IDL (interface definition language), предоставляющий базовые средства для объектно-ориентированного дизайна интерфейса. В рамках

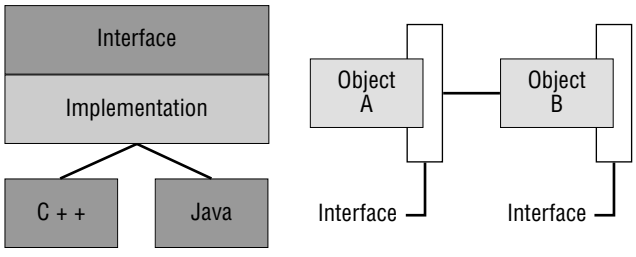

Рис. 2. Принципы абстракции на уровне интерфейса

модели для всех использующихся классов реализованы интерфейсы на этом языке.

Однако написанные интерфейсы не имеют конечного применения в сетевом приложении. Необходимо выбрать так называемый ORB. (Object Request Broker) – вспомогательный объект, который берёт всю работу по общению с сервером на себя. С одной стороны, ORB работает с нашим приложением, с другой, – с сервером, на котором хранятся разделяемые сервером и клиентом объекты.

В данной работе использовался JacORB – один из многочисленных свободно распространяемых ORBов, который предоставляет средства для написания сетевых приложений на языке программирования Java 2. В поставляемый архив с JacORB входит средство для процедуры, называемой language mapping – это генерация кода на основе написанного IDL описания. После проведения данной процедуры у нас получился набор файлов, которые впоследствии могут использоваться сервером или клиентом. Нас интересуют файлы для клиента. После генерации и добавления нескольких классов, получилась следующая архитектура (*см. рис. 3*). Данная архитектура позволяет адаптировать объекты, приходящие по сети от SWARM-сервера к внутренним структурам нашего приложения, отвечающим интерфейсам Repast.

После написания интерфейсов на IDL необходимо спроецировать эти интерфейсы на конкретные языки программирования. Это необходимо, чтобы ORBы могли взаимодействовать со вспомогательными объектами, реализованными на конкретном языке программирования. Для этого используется процедура, называемая language mapping.

В итоге наше «клиент-сервер» приложение имеет следующую архитектуру (*рис. 3*):

На этой схеме представлено, по какому принципу могут взаимодействовать два приложения, написанные на разных языках программирования, вне зависимости от архитектуры сети, от логики сервера или клиента.

Более подробная спецификация архитектуры сервера и клиента разработана с применением языка UML (*рис. 4*). Описание классов представлено в *табл. 1.*

Схематично взаимодействие клиента и сервера представлено на *рис. 5.*

На этой схеме представлено, по какому принципу могут взаимодействовать два приложения, написанные на разных языках программирования, вне зависимости от архитектуры сети, от логики сервера или клиента. Схема взаимодействия описывается следующими пунктами:

Таблица 1

Описание классов

Название класса	Описание
RemoteWholesaler	Конечный объект, который принимается клиентом в сетевом приложении. Наследует ряд интерфейсов, необходимых для JacORB.
RemoteWholesaler Operations	Данный интерфейс генерируется основываясь исключительно на IDL файле. Здесь описываются весь набор методов, которыми должен обладать объект, передающийся по сети.
IDLEntiy, Object	Данные классы используются внутренне ORBом, их реализация нас не интересует.
WholesalerAdapter	Адаптер сетевого объекта к внутренним интерфейсам. Экземпляры этого класса используются в Repast-модели в клиентском приложении.

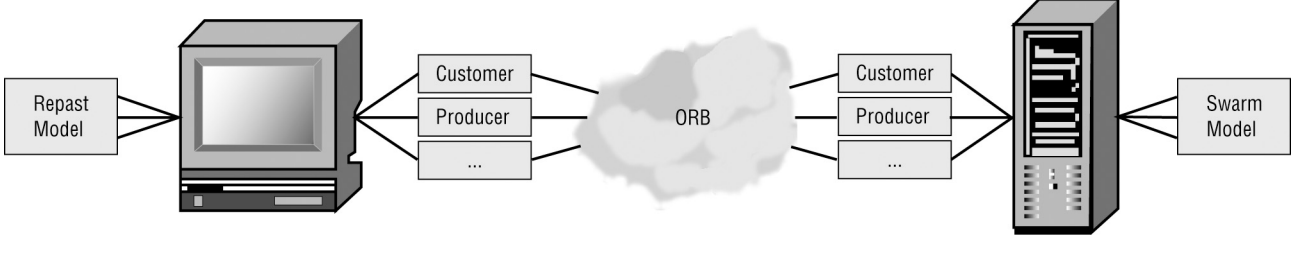

Рис. 3. Укрупнённая архитектура распределённой имитационной системы

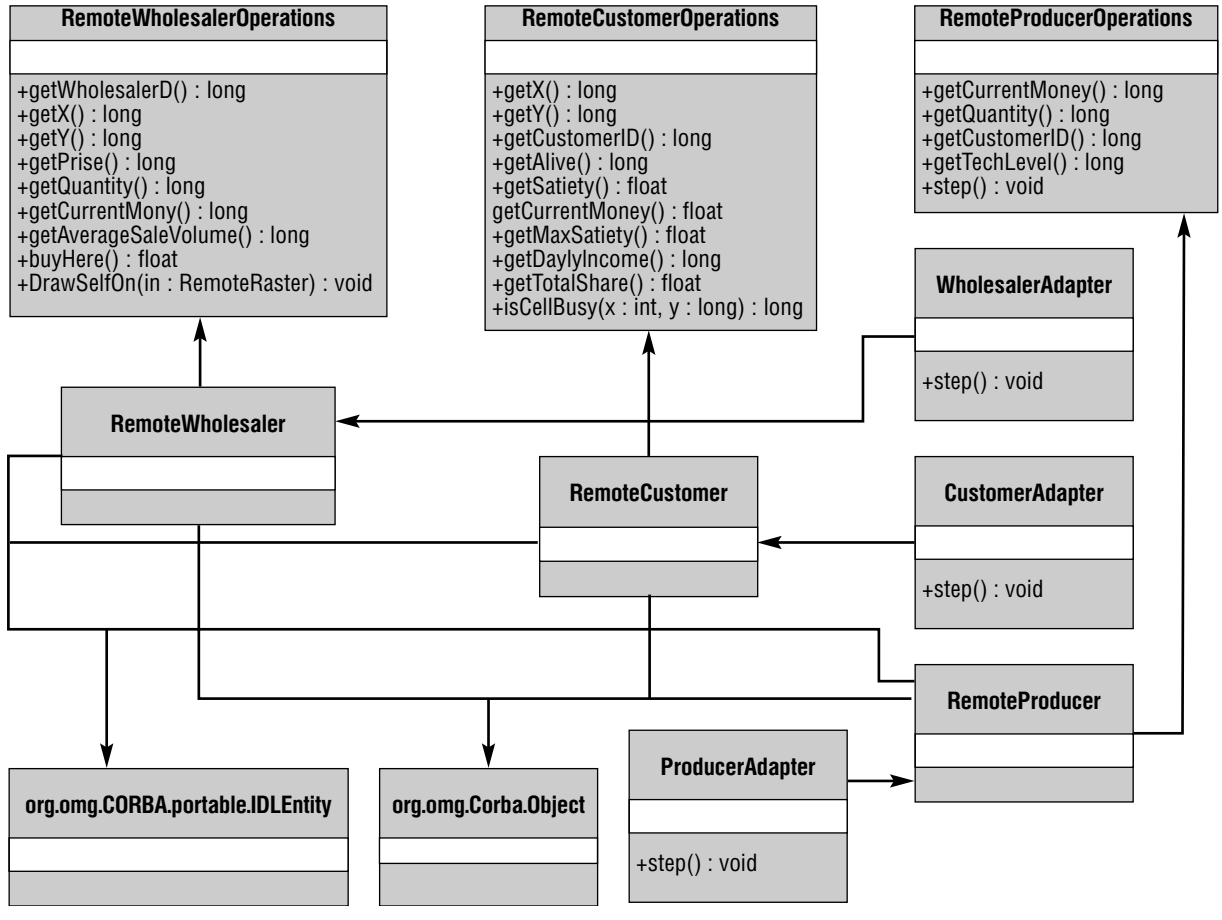

Рис. 4. Часть архитектуры клиентского приложения, отвечающая за сетевое взаимодействие

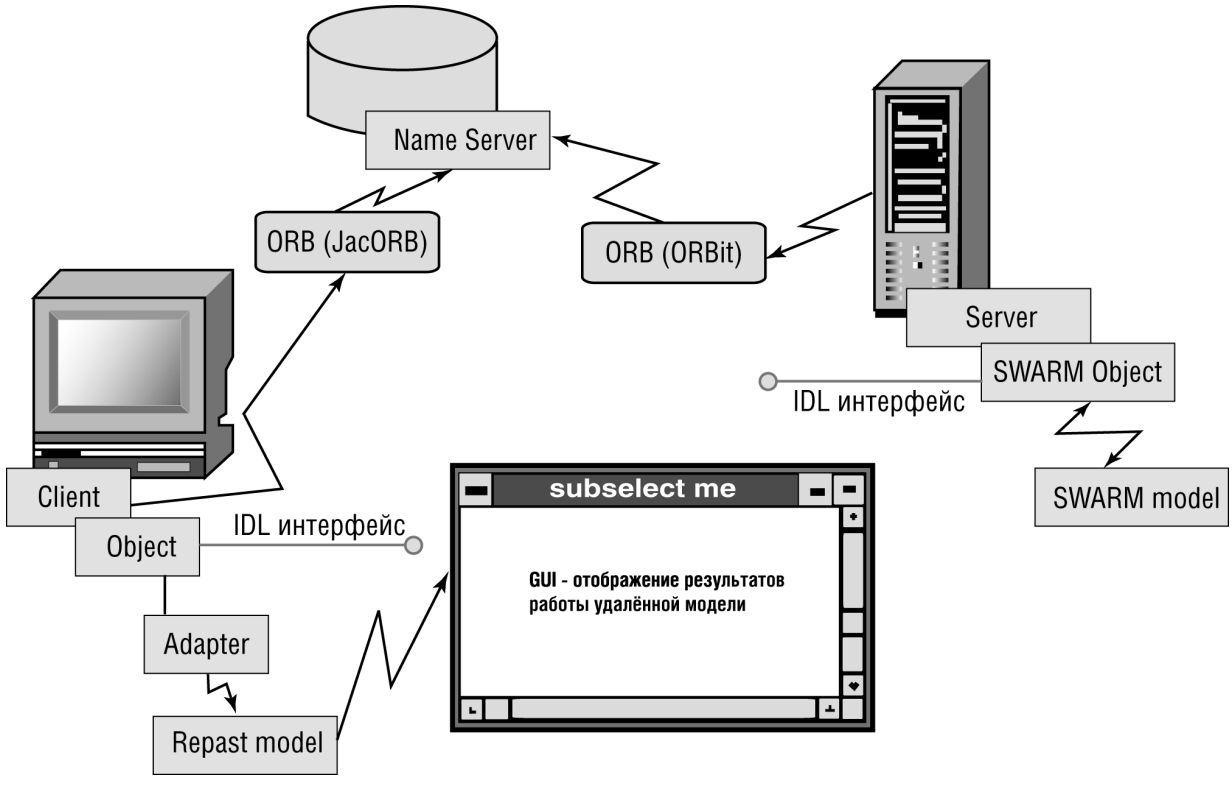

Рис.5. Взаимодействие Repast-клиента и Swarm-сервера

1. Имеется модель, нагруженная логикой. Здесь реально происходит взаимодействие моделируемых агентов по некоторым принципам. В нашем случае это модель, написанная на SWARM.

2. Все объекты модели должны предоставлять интерфейсы для ORBa – для возможности обмена данными с сервером имён. Сервер имён – своего рода база данных, где реально хранятся моделируемые агенты, и прочие объекты.

3. Клиентское приложение обращается к серверу имён с целью получить доступ к данным удалённой модели. Они поступают в это приложение в виде экземпляров классов, таких как вышеописанные RemoteWholesaler, RemoteCustomer и RemoteProducer.

4. При помощи объектов-адаптеров данные удалённой модели поступают в локальную модель. Адаптированные объекты могут использоваться локальной моделью точно так же, как если бы это были не объекты, передаваемые по сети, а обычные локальные объекты.

5. Локальная модель при помощи собственных средств отображает на экране клиентского приложения результаты выполнения удаленной модели.

Таким образом, разработаны IDL, и на их основе сгенерированы CORBA-объекты, адаптированные к новой модели для системы Repast. Новая модель Repast вместо того, чтобы брать вычислительную нагрузку на себя, обращается к ORB, который берёт данные с другой машины сети, выполняющей все вычисления.

Сбор и предварительная подготовка материалов для методического пособия по созданию имитационных моделей в системе SWARM

Данная задача – продолжение исследований в области применения многоагентных методов моделирования на базе библиотеки SWARM. В предыдущих работах создана тестовая модель социально-экономической динамики и показана применимость агентного подхода к экономическому моделированию. (Исследовательский проект 06-05-0021).

В ходе проектирования и реализации прототипов, выполнения имитационных экспериментов выявлено несколько недостатков обычного подхода, предусмотренного библиотекой SWARM. Это:

- ✧ необходимость разработки больших объёмов однообразного кода;
- ✧ описание модели в терминах, достаточно далёких от предметной области;
- ✧ для неспециалиста процесс разработки модели весьма сложен.

Очевидное решение – создание системы моделирования – позволило бы описывать модели в терминах предметной области и берущей всю работу по созданию работоспособного кода на себя.

Для описания модели должны быть разработаны специальные инструменты визуального моделирования, предполагающие работу на уровне логики модели. Примером подобного инструмента может служить редактор диаграмм с поддержкой объектов мета-мета-модели, позволяющий получить на выходе полное и непротиворечивое описание модели. Более общая задача – создание подхода к моделированию, в который вписывается широкий класс моделей.

Подобный подход – развитие модельно-ориентированного метода разработки программного обеспечения с применением специализированной социально-экономической мета-мета-модели, содержащей набор базовых концепций, в терминах которой можно определить мета-модель – описание конкретных видов объектов, возникающих в модели. Таким образом, уровень модели должен описывать лишь конкретные характеристики конкретных экземпляров объектов.

Постановка задачи

В процессе этого этапа исследований необходимо:

1. Создать систему моделирования, которая позволяла бы описывать широкий класс моделей в терминах, близких к предметной области.

2. Разработать инструменты визуального моделирования.

3. Сделать возможной генерацию полнофункционального кода по описанию модели.

Сама постановка задачи предполагает разбиение работы на три компонента – разработка подхода (мета-мета-модели), разработка инструментов моделирования и разработка средств генерации кода. Наиболее существенным представляется первый раздел. Именно здесь закладываются основополагающие принципы, на которых базируется вся дальнейшая работа.

Описание подхода

В рамках данной работы разработан подход, способный описать широкий класс моделей социально-экономической динамики. При создании исследовательского прототипа решено ограничиться базовым набором объектов.

Agents

Агенты (Agents) – базовый элемент модели. Агентом является любая сущность, обладающая

некоторыми характеристиками, состояниями и поведением. Примерами агентов могут служить покупатели, продавцы, производители и т.п.

Характеристики агентов – описание данных, которыми оперирует агент, информация о том, в каких пространствах агент обитает, информация о выполнении агентом тех или иных действий, описание алгоритма поведения агента и т.п.

Contracts

Контракты (Contracts) – мощный механизм взаимодействия агентов. Контракт – набор действий, которые должны быть предприняты «подписавшими» его агентами при осуществлении некоторых условий. Рассмотрим простейший контракт между продавцом и покупателем. Продавец предлагает контракт по приобретению некоторого товара по некоторой цене. Имеем следующие действия:

1. Покупатель должен передать продавцу определённое количество денег.

2. Если покупатель передал нужное количество денег, продавец должен передать покупателю соответствующее количество товара.

3. Если продавец не может по тем или иным причинам передать нужное количество товара, он обязан вернуть деньги.

Мы видим, что контракт состоит из набора действий, для каждого из которых можно сформулировать пред- и постусловие. Алгоритм выполнения каждого действия должен быть четко определён для соответствующего агента.

Таким образом, наша мета-мета-модель должна содержать условие (Condition), действие (Action) и контракт (Contract). Условие (Condition) – любой объект, который можно «проверить», т.е. который имеет метод ответа на вопрос: да/нет.

Areas

Необходимо также определить «физические» аспекты взаимодействия агентов. В рамках данного подхода агенты живут, двигаются и взаимодействуют в одном или нескольких пространствах (Areas). Пространство – это некоторая структура, в которой можно выделить набор позиций. На данный момент поддерживается лишь двумерное целочисленное пространство, однако существующий прототип может быть достаточно легко расширен. Пространствами могут служить графы, различные многомерные структуры и т.п.

Важно то, что агент содержит в себе всю необходимую информацию о том, как он может перемещаться в том или ином пространстве, однако непосредственным перемещением занимается глобальный объект, контролирующий корректность перемещения (см. ниже – в описании алгоритма агентов).

Resources

Ресурс (Resource) – это то, что не может возникнуть «из ниоткуда» и не может никуда исчезнуть. При взаимодействии агенты могут передавать ресурсы друг другу. Все случаи изменения общего количества некоторого ресурса (например, производство или переработка) в модели должны быть чётко определены. Таким образом, необходимо, чтобы управлением ресурсами занимался отдельный глобальный объект, могущий контролировать корректность всех операций с ними.

Описанные объекты мета-мета-модели в совокупности дают мощный подход для создания моделей. Очень широкий класс взаимодействий может быть описан в данных терминах.

Программная реализация прототипа

При разработке системы использованы следующие программные средства:

1. Среда разработки Eclipse. Кроме непосредственной разработки программного кода, которая осуществлялась в этой среде, данная система используется как платформа для создания редактора моделей.

2. Graphical Editing Framework (GEF) – plug-in для Eclipse, позволяющий создавать редакторы двумерных схем и диаграмм. Обобщенный подход, и большое количество реализованных стандартных функций позволяют без лишних затрат создавать полнофункциональные редакторы, являющиеся совместимыми (native) с Eclipse.

3. SWARM – набор библиотек для агентного моделирования. SWARM является набором библиотек, написанных на языке Objective-C, служащих основой для разработки сложных мультиагентных систем. Этот пакет в свободном доступе выложен в сети по адресу http://wiki.swarm.org.

С использованием перечисленных средств реализован программный прототип системы имитационного моделирования, работающий в среде Cygwin под управлением ОС Windows XP.

Описание системы

Разработанная система состоит из редактора моделей, генератора кода и набора шаблонных файлов. Общий вид работы системы изображён на *рис. 6.*

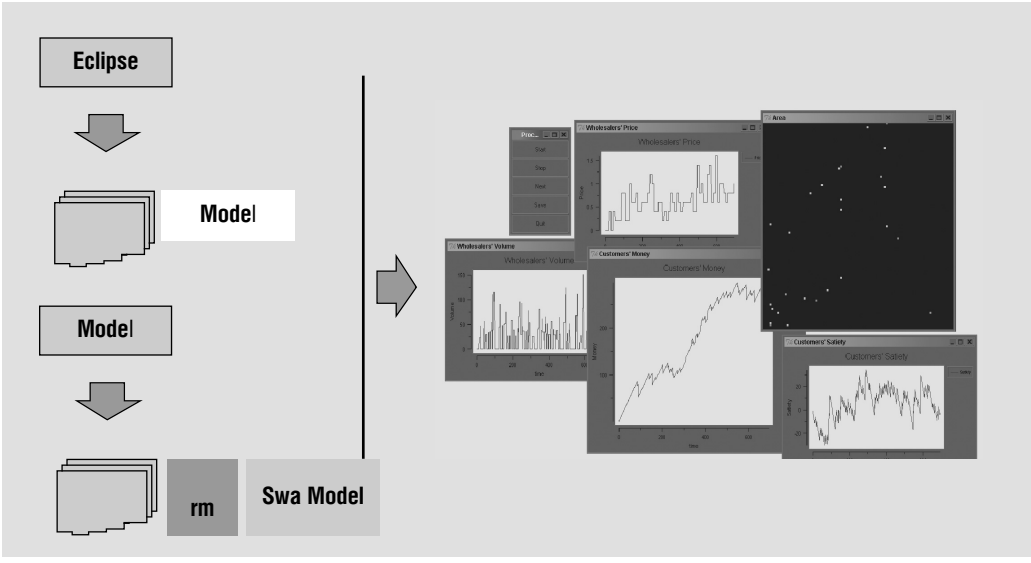

Рис. 6. Общий вид работы системы

Как видно из схемы, первый этап работы над моделью – создание описания модели в визуальном редакторе. Далее генерируется файл описания модели, который содержит исходные данные для генератора кода. После создания кода модель компилируется и запускается.

Редактор моделей представляет собой plug-in для среды разработки Eclipse (*см. рис. 7*)

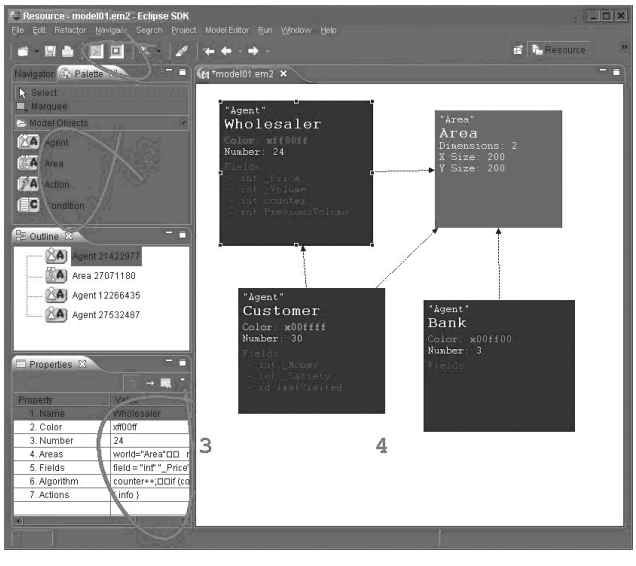

Рис. 7. Редактор моделей

На *рис. 7*:

1. Кнопки запуска модели/генерации описания.

2. Палитра объектов. Здесь представлены объекты, которые можно создавать в редакторе.

3. Редактор свойств объектов. Здесь можно изменить свойства выделенного объекта.

4. Окно редактирования, в котором отображается текущее состояние модели.

Редактор позволяет создавать следующие объекты:

✧ **Agent.** Объект Agent имеет следующие характеристики:
- ✦ Name – название объекта;
- ✦ Color – цвет, которым объект отображается при визуализации;
- ✦ Number – число объектов данного типа;
- ✦ Areas – информация о том, в каких пространствах данный агент обитает.

✧ **Атрибуты:**
- ✦ **mobility** – подвижность агента (какое максимальное расстояние он может пройти за один ход);
- ✦ **sight** – поле зрения агента, т.е. на какое расстояние он видит.

Зависимость агента от пространства визуализируется также соединяющей линией.

✧ **Fields** – поля данных агента. Вводятся в формате.

Поле имеет тип и имя. Поля отображаются также на схеме. Если имя поля начинается с подчёркивания, это сигнал генератору, что необходимо построить аггрегированный график по всем агентам данного типа. Кроме того, автоматически создаются методы доступа к полям, например, для поля Money создадутся методы getMoney, setMoney, incMoney и decMoney.

✧ **Algorithm** – алгоритм поведения агента, который вызывается на каждый такт.

Пример:

```
Money++;
Satiety--;
if (lastVisited == nil)
lastVisited = [@Area getAnyWholesalerNot: nil];
if ([@Area near: self To: lastVisited])
{
if (Satiety < 0)
{
int amount = (-Satiety + 100);
int price = [lastVisited getPrice];
[lastVisited incVolume: amount];
Satiety = 100;
Money -= price * amount;
}
lastVisited = [@Area getAnyWholesalerNot: nil];
}

[@Area go: self To: lastVisited];
```

В описании алгоритма могут использоваться глобальные объекты, а также поля объекта.

- **Actions** – информация о том, как данный агент выполняет действия (Actions). На данный момент не поддерживается.

- **Area.** Объект Area имеет следующие характеристики:
  - Name – название объекта.
  - Dimensions – размерность пространства (на данный момент поддерживаются только двумерные пространства).
  - Display – включена или отключена визуализация пространства.
  - Tangibility – «осязаемость» пространства. В частности, могут ли два объекта находиться в одной точке.
  - Distance – способ расчёта расстояния. На данный момент поддерживается только евклидова мера.
  - X Size, Y Size – размеры пространства

- **Action.** Объект Action имеет следующие характеристики:
  - Name – название объекта.
  - Participants – информация об участниках.

- **Condition**
  - Name – название объекта.
  - Participants – информация об участниках.
  - Value – возвращаемое значение.

После создания модели можно сгенерировать файл описания модели.

Запускается модель, генератор, создаётся программный код. Он компилируется и запускается. Если на каком-то этапе происходит сбой, выводится сообщение об ошибке.

Выходные данные у редактора моделей – это файл описания. Далее начинает работу **генератор кода.** Эта система устроена следующим образом. Вначале считывается файл описания модели. Вся информация о модели сохраняется в специальных структурах (Data Storage). После этого шаблонные файлы дополняются необходимым кодом. **Шаблонные файлы** – код на Objective C с пометками, указывающими генератору, куда необходимо вставить недостающие куски кода. Это осуществляется объектом Incutter. Генератор реализован на языке C. Общий вид работы генератора кода представлен на *рис. 8.*

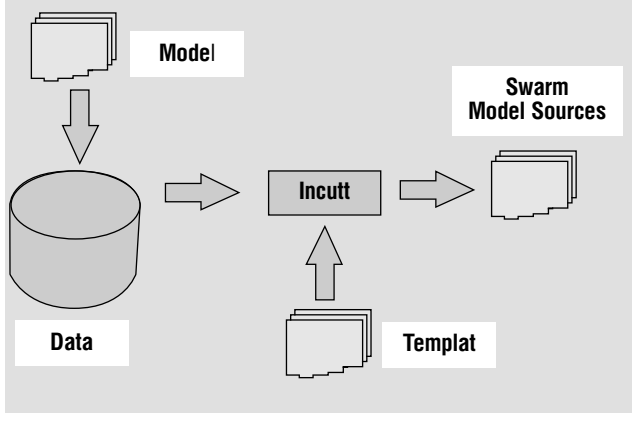

Рис. 8. Работа генератора кода

Дальнейшее развитие

Разработанный прототип не поддерживает ряд концепций, предполагаемых подходом. Однако он достаточно легко расширяемый. Наиболее перспективные пути усовершенствования системы:

1. Расширение и усложнение методов, предоставляемых глобальными объектами, чтобы они могли взять на себя бóльшую часть рутинной работы, предоставляя возможность работать непосредственно над логикой модели.

2. Внедрение новых глобальных объектов, помогающих при моделировании.

3. Разработка более гибкого подхода к взаимодействию агентов, например, чтобы агенты могли «запоминать» информацию о конкретных экземплярах других объектов.

4. Реализация всех остальных концепций, предполагаемых подходом – контракты, действия, ресурсы.

Результаты

1. Разработана распределённая архитектура системы иммитационного моделирования. Сервером является система SWARM реализованная на языке Objective C, а клиентское приложение реализовано в системе Repast на языке Java.

2. Реализованы алгоритмы интеграции и кластеризации знаний на основе онтологий.

3. Разработаны система моделирования, для описания моделей в терминах предметной области и инструменты визуального моделирования. Реализована генерация полнофункционального кода по описанию модели. Разработанная система существенно облегчает разработку моделей с использованием SWARM. Код, сгенерированный данной системой, можно использовать в качестве основы для дальнейшей модернизации и создания более сложных моделей. Несмотря на то, что реализованы не все концепции, главная задача выполнена – реализованная система позволяет абстрагироваться от низкоуровневых аспектов моделирования и сконцентрироваться на разработке логики модели.

Выводы

Разрабатывается одно из наиболее перспективных направлений в теоретических и прикладных исследованиях сложных социальных систем – многоаспектное математико-информационное моделирование, воплощённое в форме распределённых имитационных систем на основе парадигмы взаимодействия индивидуальных сущностей (individual-based systems). Построена единая информационно-вычислительная система, предоставляющая удалённым пользователям удобный интерфейс для самостоятельного расширения возможностей взаимодействия с системой, обладающая развитыми средствами автоматизированной разработки и анализа имитационных многоаспектных моделей.

Задача, решаемая в данном проекте, состоит в построении методологии моделирования и программной архитектуры распределённой информационно-вычислительной системы для генеративного описания и изучения многофакторных моделей социальной динамики. Эти функции эффективно реализованы на основе развития принципов многоагентного моделирования и средств распределённого программирования в стандарте CORBA. Критически проанализированы архитектура распределённых вычислений CORBA и технология мобильных программных агентов с их последующей специализацией для решения задач имитационного моделирования динамических процессов в сложных социально-экономических системах. Результатом является достижение следующих целей:

- ✧ создание новой методологии построения распределённых систем имитационного моделирования на основе совместного применения CORBA и многоагентных систем;
- ✧ разработка новых математических моделей и алгоритмов масштабируемых вычислительных экспериментов на моделях социальной динамики в локальных сетях. ■

Работа выполнена при финансовой поддержке гранта РФФИ 07-07-00058.

Литература

1. А.А. Шутов, Г.А. Баутин, И.В. Куркина. Реализация экономической модели в распределенной системе моделирования. // Известия АИН им. А.М. Прохорова. Бизнес информатика. 2005. Т.12.
2. Object Management Group // www.omg.org
3. Common Object Request Broker Architecture // материалы с сайта www.omg.org
4. ORBit // материалы с сайта http://orbit-resource.sourceforge.net/ и http://www.gnome.org/projects/ORBit2/
5. Recursive Porous Agent Simulation Toolkit // материалы с сайта http://repast.sourceforge.net
6. Swarm Development Group http://www.swarm.org
7. R. Andrew McCallum The GNU Objective-C Language Manual http://theory.uwinnipeg.ca/gnu/libobjects/objective-c_toc.html
8. http://homepage.mac.com/charlesgunn/oorangeDoc/interpretedOC/interpretedOC.html
9. The Objective-C Programming Language, 2002, 2004 Apple Computer, Inc. http://developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC/ObjC.pdf
10. Чепурин М.Н., Кисилева Е.А. Курс экономической теории –М.: АСА, 2002.
11. Сидорович А.В. и др. Курс экономической теории. Общие основы экономической теории. Микроэкономика. Макроэкономика. – М.: ДИС, 2001.