Многоагентная платформа для создания распределенных приложений

Э.В. Шогулин, В.В. Андреев

Целью данной работы являются исследования в области разработки нового типа программного приложения такого, как многоагентная система, и разработка платформы для возможности их реализации. В результате реализована платформа для решения разнообразных научных, допускающих распараллеливание, задач. Также на базе платформы реализована многоагентная система для бизнес-приложений.

1. Введение

На сегодняшний день существует множество характеристик, используя которые мы можем разбить различные программные приложения на несколько типов. Так, можно разбить приложения по степени интерактивности с пользователем - приложения с графическим интерфейсом, позволяющие взаимодействовать с пользователем, так и без графического интерфейса. Примерами приложениями без графического интерфейса являются так называемые сервисы (service) в операционной системе Windows или daemon в операционной системе Linux. Можно также разбить приложения по взаимодействию различных звеньев (tier) приложения. Существуют приложения, все звенья которых представляют одно целое и расположены на одном компьютере. И существуют приложения, звенья которых расположены отдельно и часто на различных компьютерах. Приложения, звенья которых выполнены отдельно друг от друга, называются распределенными. Существуют Веб-приложения, работать с которыми пользователь может только через Интернет-браузер.

Согласно [1] разработкой средств поддержки создания многоагентных систем занимается большое количество коллективов в мире, но, как правило, эти работы носят теоретический характер. Целью данной работы является описание нового типа такого программного приложения, как многоагентное приложение. Провести исследования в этой области и реализовать платформу для создания приложений этого типа. На сегодняшний день на базе созданной платформы реализованы многоагентное приложение для решения научных задач и бизнес приложение.

2. Теоретическая часть

Многоагентное приложение является распределенным приложением с возможностью взаимодействия с пользователем. Возможность взаимодействия с пользователем расширяет круг возможных задач, которые могут быть решены приложением этого типа.

Программная реализация многоагентного приложения кардинально отличается от написания других типов программ. Здесь основной сущностью является агент и, следовательно, создание приложения заключается в написании агента. На текущий момент существует множество определений агента. Авторы придерживаются определения агента и его свойств, выделенных в работе [3]. Многоагентная платформа прозрачна для разработчика алгоритма вычисления. Проанализировав требования к платформе и наиболее распространенные из существующих платформ, авторы при выборе программной платформы остановились на технологии Microsoft .NET. Реализованная многоагентная платформа позволяет использовать для написания агента любой .NET язык. Для реализации агента достаточно реализовать класс, пронаследованный от базового класса агентов Agent, и перегрузить необходимые методы.

На рис.1 показан простейший агент, выполняющий параллельно три свои части на различных компьютерах. Для портирования агента разработчик агента должен использовать метод StrartThread базового класса агента Agent. В качестве входного аргумента подаётся экземпляр делегата RunFunctionHandler, объявленного во внутренних библиотеках платформы. В свою

очередь, делегат принимает указатель на метод, который будет выполнен после портирования агента. Так Function1 и Function2 выполняются на других компьютерах после портирования агента. Перед началом их выполнения агент портируется на эти компьютеры и после выполнения агент, вызвавший их, получает результат их работы.

Функционирование такого многоагентного приложения происходит следующим образом. Платформа создает экземпляр агента и запускает метод Run созданного экземпляра на некотором компьютере. Сразу же происходит портирование экземпляра агента на другой компьютер для запуска метода Function1. После портирования платформа продолжает выполнять код метода Run исходного агента. Параллельно этому процессу на другом компьютере запускается и начинает выполняться метод Function1 у портированного ранее экземпляра агента. Так же метод Function2 выполняется на другом компьютере параллельно методу Run и методу Function1. Платформа предоставляет возможность синхронизации распараллеленных агентов и воспользоваться результатом их работы. Это делается путем вызова метода Wait.

```
public class CalculationAgentWithSleeping: Agent
{
   private Object Function1(Object param)
   {
      Thread.Sleep(80000);
      return "Function1 result";
   }
   private Object Function2(Object param)
   {
      Thread.Sleep(60000);
      return "Function2 result";
   }
   private Object Function3(Object param)
   {
      Thread.Sleep(40000);
      return "Function3 result";
   }

   public override Object Run(Object param)
   {
      FunctionEvent functionEvent1 = StartThread(
         new RunFunctionHandler(Function1));
      FunctionEvent functionEvent2 = StartThread(
         new RunFunctionHandler(Function2));
      Object functionEvent3Value = Function3(null);

      Wait(functionEvent1, functionEvent2);

      return
         Convert.ToString(functionEvent1.Value) +
         Convert.ToString(functionEvent2.Value) +
         Convert.ToString(functionEvent3Value);
   }
}
```

Рис. 1. Исходный код агента, распараллеливающего задачи

Таким образом, частный случай использования многоагентной платформы является написание приложений для распараллеливания вычислительных ресурсоемких задач на различные машины в глобальной сети. Для этого была разработана управляемая оболочка для различных математических пакетов, пригодная к использованию многоагентными приложениями, разработанных на базе описываемой платформы.

Кроме портирования, при котором происходит распараллеливание одного экземпляра агента, платформа позволяет портировать экземпляр перемещением. В частности, такой вариант предложен [2]. Тип портирования может изменяться программно во время выполнения агента.

Портирование обычных приложений на многоагентную платформу позволяет использовать все преимущества многоагентного приложения без перекомпиляции приложения. Исходные коды приложения не требуются. Для реализации портировании приложения меняется только та его часть, которая работает с потоками.

Допустим, имеется класс. На рис.2 приведена его часть, которая работает с потоками. После портирования код этой части будет выглядеть как на рис.3. При этом портирование происходит в автоматическом режиме с сохранением текущей функциональности.

Основное отличие сгенерированного кода от первоначального, заключается в автоматическом создании метода, который и запускается параллельно на другом хосте. В качестве результата выполнения, вместо использования члена этого же класса для возврата результата в оригинальном коде, используется результат выполнения сгенерированного метода.

```
public class CalculationFunction
{
   public Object Value;
   public Object Result;
   public void Calculate()
   {
      Thread.Sleep(60000);
      Result = "some parallel result";
   }
}

public class CalculationAlgorithm
{
   private Object Value;
   private Object Result;
   private void Calculate()
   {
      Thread.Sleep(40000);
      Result = "some serial result";
   }

   public Object Run(Object param)
   {
      CalculationFunction calculationFunction =
         new CalculationFunction();
      calculationFunction.Value = "some parallel value";

      Thread thread = new Thread(
         new ThreadStart(calculationFunction.Calculate));
      thread.Start();

      Value = "some serial value";
      Calculate();

      thread.Join();
      return
         Convert.ToString(calculationFunction.Result) +
         Convert.ToString(Result);
   }
}
```

Рис. 2. Оригинальный исходный код для распараллеливания

На какой именно компьютер будет портирован агент, определяется платформой во время выполнения агента в соответствии со следующими критериями: качество сетевого соединения, производительность хоста, загруженность хоста, надежность хоста.

```
public class CalculationFunction
{
   public Object Value;
   public Object Result;
   public void Calculate()
   {
      Thread.Sleep(60000);
      Result = "some parallel result";
   }
}

public class CalculationAlgorithm : Agent
{
   private Object GeneratedFunction45E4(Object param)
   {
      CalculationFunction calculationFunction =
         (CalculationFunction)param;
      calculationFunction.Calculate();
      return calculationFunction;
   }

   private Object Value;
   private Object Result;
   private void Calculate()
   {
      Thread.Sleep(40000);
      Result = "Function3 result";
   }

   public override Object Run(Object param)
   {
      CalculationFunction calculationFunction =
         new CalculationFunction();
      calculationFunction.Value = "some parallel value";

      FunctionEvent functionEvent = StartThread(
         new RunFunctionHandler(GeneratedFunction45E4),
         calculationFunction);

      Value = "some serial value";
      Calculate();

      Wait(functionEvent);
      return
 Convert.ToString(((CalculationFunction)functionEvent.Value).Result) +
         Convert.ToString(Result);
   }
}
```

Рис. 3. Результирующий код после автоматической генерации

2.1 Платформа для построения бизнес-приложений

Для большинства бизнес приложений мы можем выделить некоторую общую функциональность, задача которой заключается в предоставлении пользователю возможность сохранять, читать и отображать некоторые сущности, такие как, например: клиент, счет, заказ, договор и так далее. Эти сущности, как правило, известны под названием бизнес-сущности. Типичное бизнес приложение, как правило, имеет трехзвенную архитектуру: клиент, сервер приложений, СУБД. При этом каждое звено делится на слои. Например, типичный сервер приложения имеет следующие слои: слой доступа к хранилищу (data access layer); слой работы с объектами хранилища (data entity layer); слой, содержащий бизнес логику (business logic layer). Раз-

личные звенья в бизнес-приложениях связывают потоки данных. Каждое из звеньев уникально и выполняет отведенную ему роль. Если же бизнес приложение выполнено на многоагентной платформе, то её архитектура будет кардинально отличаться. Основные отличия заключаются в статичности компонентов приложения в случае типичной реализации и в динамике компонентов приложения в случае реализации на базе многоагентной платформы. Рассмотрим многоагентную платформу в работе. При вводе данных формируется агент, который выполняет свою работу на клиентском компьютере. Например, здесь может осуществляться первичная проверка введенных данных с целью уменьшения сетевого трафика. Если данные верны, то агент перемещается на сервер приложений. Здесь происходит основная работа агента. После отработки агента на сервере приложений агент, согласно бизнес логике, сохраняется и возвращается клиенту. После возвращения агента клиентское приложение может отобразить изменившегося агента.

Итак, подчеркнем наиболее значимые отличия реализации бизнес-приложений как многоагентное приложение от обычного распределенного и выделим преимущества:

- скрытость логики связи между отдельными звеньями (частями) приложения;
- скрытость протокола;
- реализация всей логики работы с сущностью в одном типе;
- простота и скорость разработки.

Выделим также и недостатки:

- отсутствие явного разделения логики агента;

При портировании агента осуществляется перенос не только данных (свойств агента), но и кода (код агента). Это позволяет выполнять код заказчика на мощностях исполнителя. При этом многоагентная платформа обладает инфраструктурой, необходимой как для заказчика (управление агентами), так и для исполнителя (оптимальное распределение агентов между хостами, система безопасности).

3. Реализация многоагентной платформы

Перед разработкой многоагентной платформы были изучены наиболее распространенные платформы из существующих согласно работе [4], и была поставлена задача создать платформу максимально понятной как для рядового разработчика, так и исследователя многоагентных систем. Также разработчики уделили внимание не только внешним интерфейсам, но и внутренней открытости платформы. При этом гибкость архитектуры не должна быть реализована за счет потери производительности. Программная платформа .NET, на которой была построена рассматриваемая многоагентная платформа, является компонентно-ориентированной. Отчасти и за счет этого разработчикам удалось добиться преследуемых целей. На рис.4 приведены основные компоненты разработанной многоагентной платформы и их взаимодействие. Стрелками показаны взаимодействия различных компонентов. Направление стрелок указывает направление взаимодействия.

При каждом этапе функционирования агента происходит последовательное движение его экземпляра от одной очереди однотипных компонентов платформы до другой. В рамках однотипной очереди компонентов содержаться разные компоненты многоагентной платформы решающие одну и ту же задачу. В каждой очереди присутствует как минимум один компонент добавленный разработчиками платформы. Кроме этого компонента каждая очередь может содержать неограниченное количество компонентов этого же типа разработанных сторонними разработчиками.

Реализованная многоагентная платформа включает в себя следующие приложения и сервисы:

- Контролирующее приложение (Controlling Application). Под контролирующем приложением понимается приложение, позволяющее пользователю взаимодействовать с другими частями платформы и, в частности, управлять агентами.

- Хост сервис (Host Service). Под хост сервисом понимается компонент многоагентной платформы, предназначенный, прежде всего, для запуска агентов на компьютере, где он установлен.

- Сервис для управления хостами (Host Distributor Service). Под сервисом для управления

хостами понимается компонент многоагентной платформы, предназначенный для предоставления информации о доступных хост сервисах и для портирования агентов.

- Сервис для управления сборками агентов (Assembly Manager Service).
- Сервис для управления экземплярами агентов (Instance Manager Service).
- Сервис сбора информации об окружающей среде (Environment Collector Service).

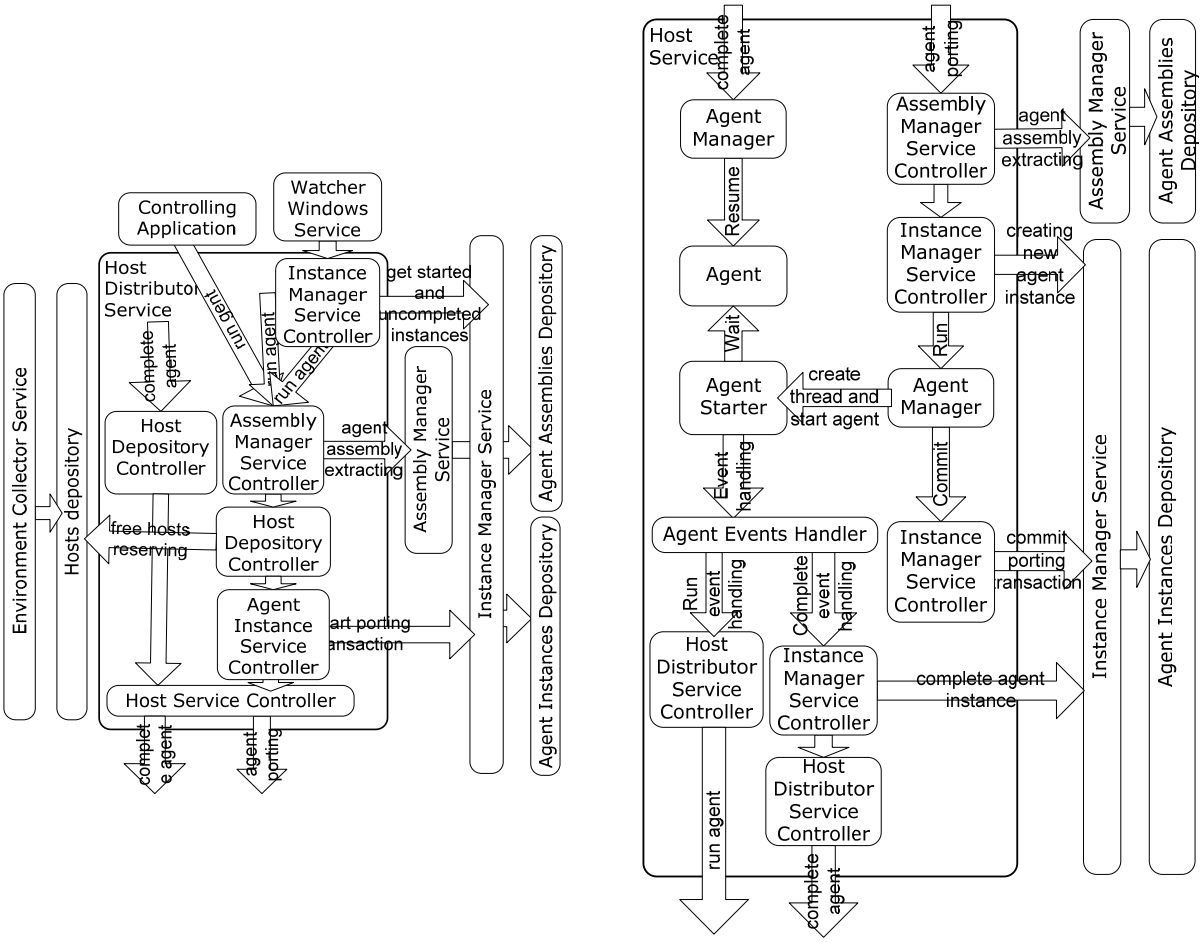

Рис. 4. Архитектура многоагентной платформы

4. Заключение

В работе реализована многоагентная платформа, на основе которой предложен подход создания программных приложений нового типа. В работе описаны области, в которых могут использоваться эти приложения. Дальнейшее развитие работы связанно с изучением поведения агентов и исследованиями с целью добиться реализации оптимального портирования агентов и реализации оптимального их взаимодействия друг с другом и с внешней средой.

Литература

1. Городецкий В.И., Карсаев О.В., Конюший В.Г., Самойлов В.В., Хабалов А.В. Среда разработки многоагентных приложений MASDK. // Информационные технологии и вычислительные системы. № 1-2, 2003. -С. 26-41.
2. Нили М. Создание мобильных агентов .NET для взаимодействия по сети // MSDN Magazine Русская Редакция. Февраль 2006.
3. Латынцев А.А. Агент системы анализа сложных изображений // Диссертация на соискание ученой степени кандидата технических наук. -Красноярск, 2006. -С. 6-16.
4. Бакиров Т.К. Автоматизированная система анализа защищенности корпоративной вычислительной сети на основе многоагентного подхода. // Диссертация на соискание ученой степени кандидата технических наук. -Уфа, 2006. -С. 147-151.