Лабораторная работа № 3.
Коммуникационная подсистема. Язык общения агентов ACL

Содержание работы:

Цель работы:

1. Закрепить теоретические знания о языке общения агентов ACL.
2. Получить практические навыки создания общающихся между собой агентов.

1. Теоретические сведения

1.1. Коммуникационная подсистема

Основной контейнер JADE поддерживает реестр RMI, используемый другими агентными контейнерами для своей регистрации в агентной платформе. Также основной контейнер хранит таблицу всех контейнеров с объектными ссылками на их RMI. Кроме того, он хранит таблицу Глобальных Агентных Дескрипторов (Agent Global Descriptor Table), где каждому имени агента поставлены в соответствие данные своей AMS и объектная ссылка RMI своего контейнера.

Когда запускается основной контейнер, он создает внутренний реестр RMI на текущей ЭВМ, слушающий указанный TCP/IP порт. Затем он запускает системных агентов FIPA (ACC, AMS и DF).

Когда запускается на исполнение новый контейнер, он ищет реестр RMI в основном контейнере и получает от него объектную ссылку на основной контейнер. Новый контейнер регистрирует себя в основном контейнере. Информация о новом контейнере записывается в Agent Container Table (таблицу агентных контейнеров). Затем новый контейнер сообщает основному, когда запускается или уничтожается агент, таким образом обновляется таблица Agent Global Descriptor Table.

Для повышения производительности каждый контейнер кэширует объектные ссылки на другие контейнеры, когда получает от них сообщение. Таким образом удается избежать обращения к Agent Global Descriptor Table каждый раз при отправке сообщения. Тем не менее, поскольку новые контейнеры могут появиться и старые могут быть удалены, при возникновении исключения при обращении к контейнеру реализуется новый запрос в Agent Global Descriptor Table.

При отправке сообщения в JADE возможны следующие ситуации:

1. Если получающий агент «живет» в том же контейнере, что и отправляющий, то объект Java, представляющий ACL сообщение, передается получателю в виде объекта-события, без какого-либо преобразования сообщения (например, как показано на рис. 1, Агент 1 отправляет сообщение Агенту 2).

2. Если получающий агент «живет» на той же платформе JADE, но в другом контейнере, то ACL сообщение посылается путем RMI. Java RMI позволяет «прозрачное» транспортирование объектов, и этим удается избежать преобразования сообщения. Получающий агент также получит объект Java, как и в первом случае (см. рисунок 1, Агент 3 посылает сообщение Агенту 2. В этом случает объектная ссылка на контейнер Агента 2 получается из кэша, и нет надобности обращения к основному контейнеру агентной платформы).

3. Если получающий агент находится на другой агентной платформе, то стандартный протокол IIOP и интерфейс OMG IDL применяются, согласно стандарту FIPA. Также, имеется возможность подключить другие протоколы – HTTP, bit-efficient (прилагаются как дополнение к JADE) или другие, реализованные пользователем. В таком случае ACL сообщение преобразуется в стороковый формат. На принимающей стороне строковый формат ACL сообщения преобразуется в объект Java типа String, а затем в в объект Java (встроенный в

JADE) типа ACLMessage. Далее, этот объект будет передан принимающему агенту, используя методы Java-событий или вызовов RMI.

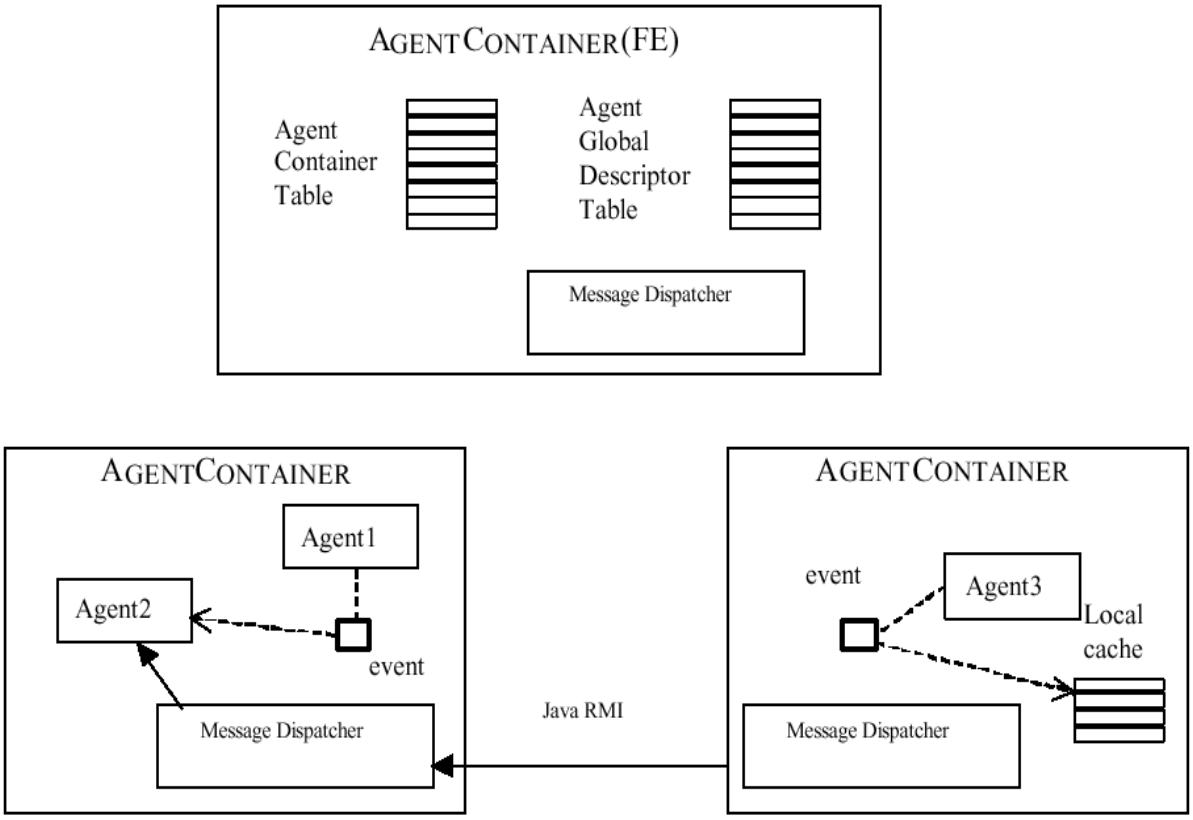

Рис. 1 - Коммуникация между агентными контейнерами

JADE позволяет таким образом выбрать наиболее эффективный способ пересылки сообщений, в зависимости от месторасположения получающего агента.

Для «внешнего мира» агентная платформа имеет единственный интерфейс – агент ACC (Agent Communication Channel, коммуникационный канал агентной платформы). Этот агент поддерживает протокол CORBA IIOP и «слушает» удаленные вызовы. Каждый раз, получая ACL сообщение в формате строки, он распознает это сообщение и преобразует его в Java-объект класса ACLMessage, используемый JADE-агентами. Реализуется и обратная операция при отправке ACL сообщения с JADE-платформы на не-JADE-платформу. Таким образом, реализуется принцип, согласно которому конкретный агент не знает о коммуникационных механизмах между агентами. В самом деле, каждый агент является отдельным самостоятельным классом, исполняющимся в собственном потоке (Thread).

1.2. Язык общения агентов - FIPA ACL.

Для общения интеллектуальных агентов разработаны специальные языки (Agent Communication Languages - ACL), основанные на теории речевых актов. В данном цикле работ используется язык FIPA ACL, являющийся основным языком для FIPA. Сообщения на этом языке содержат следующие основные поля:

```
(<Тип сообщения>
  :sender          <Отправитель сообщения>
  :receiver        <Получатели сообщения>
  :content         <Содержание сообщения>
  :reply-with      <Метка для исходящего сообщения>
  :in-reply-to          <Ссылка на входящее сообщение>
  :replyBy         <Лимит времени на ответ>
  :language        <Язык сообщения>
  :ontology        <Онтология>
  :protocol        <Используемый протокол>
  :conversation-id <Идентификатор разговора>
)
```

Типы сообщений языка FIPA ACL, соответствующие различным речевым актам, представлены в таблице Б.1 приложения Б.

Параметр :sender содержит идентификатор (AID) агента, посылающего сообщение. В параметре :receiver указывается множество идентификаторов агентов – получателей сообщения. Поле :content содержит собственно содержание сообщения, записанное на некотором языке, указанном в поле :language. В данном цикле лабораторных работ в качестве языка содержания будет использоваться язык FIPA-SL0, предложенный и развиваемый FIPA (спецификация FIPA00008). Изучению этого языка посвящена следующая лабораторная работа.

Параметр :reply-with содержит выражение, которое будет использоваться отвечающим агентом для идентификации данного (входящего) сообщения. Это необходимо, когда агент одновременно ведет несколько диалогов. В параметре :in-reply-to указывается ссылка на сообщение, ответом на которое является данное сообщение. Поле :ontology содержит имя онтологии, используемой в содержании сообщения. Под онтологией понимается явное формальное определение всех элементов языка предметной области.

Поле :protocol содержит идентификатор протокола, используемого посылающим сообщение агентом. Это дает дополнительную информацию для интерпретации сообщения.

Параметр :conversation-id содержит выражение, которое используется для идентификации текущей последовательности коммуникативных актов, составляющих разговор. Этот параметр может использоваться агентом для управления его стратегиями общения и действиями. Кроме того, он может обеспечивать дополнительный контекст для интерпретации смысла сообщения.

1.2. Реализация общения агентов в JADE. Класс **ACLMessage**.

В соответствии со спецификацией FIPA, агенты общаются посредством асинхронной посылки сообщений, в которых передаваемая информация представляется объектами класса **ACLMessage**. Типы речевых актов языка FIPA ACL представлены в классе ACLMessage статическими константами целого типа, например: ‘INFORM_IF’, ‘QUERY_REF’, ‘REQUEST’ и т.д. Тип создаваемого сообщения может задаваться в параметре конструктора сообщения:

```
ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
```

Другой способ задать тип сообщения – использовать в теле программы метод setPerformative:

```
msg.setPerformative(ACLMessage.CONFIRM);
```

Метод **Agent.send()** позволяет выполняет пересылку сообщения агентам-адресатам, идентификаторы которых указаны в поле :receiver. Данный метод скрывает от пользователя все детали транспортировки сообщений, независимо от местонахождения агентов-получателей (локального или удаленного).

Перечень методов класса **ACLMessage** приведен в таблице Б.2 приложения Б.

Для отправки агентом требуемого сообщения необходимо в теле метода **action()** соответствующего поведения реализовать следующую последовательность действий:

1. Создать новый объект - экземпляр класса **ACLMessage**.

2. Для созданного объекта msg установить требуемые значения всех полей ACL-сообщения. Значения полей могут устанавливаться с помощью соответствующих методов setLanguage(), setContent(), setOntology() и т.п., либо вводиться пользователем с клавиатуры и добавляться к сообщению.

3. Послать сообщение с помощью метода send(msg).

1.3. Пример агента посылающего сообщение.

Пример реализации агента, имеющего единственное поведение, которое создает и посылает одно сообщение к DummyAgent, приведён в приложении А.

В данном примере в методе setup() создается и добавляется агенту единственное поведение – объект b, класс которого myBehaviour реализуется как расширение базового класса SimpleBehaviour. В методе action() этого поведения создается новое сообщение msg типа INFORM. Имя агента-получателя сообщения вводится пользователем, для чего используется буферизованный входной поток buff. Если пользователь не вводит имя (пустая строка), то по умолчанию адресатом является DummyAgent с именем ‘da0’, который должен быть запущен на платформе. Создается уникальный идентификатор receiverAID агента-получателя, который затем передается в качестве параметра методу addReceiver(), добавляющему в сообщение значение слота receiver. Затем в сообщении последовательно устанавливаются значения слотов language, encoding, conversation-id и content, после чего сообщение отправляется получателю (методом send(msg)).

В методе done() переменной finished присваивается значение ‘true’, которое сразу же возвращается этим методом. В результате данное поведение будет выполнено только один раз.

1.4. Обработка получения сообщения. Класс **MessageTemplate**

Для того чтобы получить сообщение от другого агента необходимо вызвать метод **Agent.receive()**. Данный метод останавливает поведение агента до тех пор пока не будет получено сообщение от другого агента. Данный метод работает нормально до тех пор пока у агента существует одно поведение, но в случае если агент реализует несколько взаимодействий одновременно возникает необходимость различать входящие сообщения. Изначально все входящие сообщения поступают в одну общую очередь входящих сообщений агента. С помощью класса **MessageTemplate** можно создать шаблон, который позволяет выбирать подходящие сообщения из очереди входящих сообщений. Класс **MessageTemplate** позволяет наложить условие при получении сообщения на любое поле класса **ACLMessage**. Эти условия могут быть скомбинированы в более сложный правила с помощью логических операторов. Конструктор для класса

MessageTemplate применяется только в том случае, если есть необходимость задать нестандартные условия на получаемое сообщение (в данном цикле лабораторных работ этот вопрос не рассматривается).

Условия задаются с помощью методов, например для того, чтобы получать только информационные сообщения пишется следующая строка:

```
MessageTemplate mt = MessageTemplate.MatchPerformative
                                    (ACLMessage.INFORM);
```

Метод **Agent. receive(MessageTemplate)** позволяет выполняет получение сообщения с заданными параметрами

Перечень методов класса **MessageTemplate** приведен в таблице Б.3 приложения Б.

1.5. Пример агента получающего сообщение.

Пример реализации агента, имеющего единственное поведение, которое получает сообщение типа INFORM на языке human-language от любого агента, приведён в приложении А.

В данном примере в методе setup() создается и добавляется агенту единственное поведение – объект b, класс которого myBehaviour реализуется как расширение базового класса SimpleBehaviour. В методе action() этого поведения создается условия на приём сообщения, после чего сообщение принимается(методом receive(tm)).

В методе done() переменной finished присваивается значение ‘true’, которое сразу же возвращается этим методом. В результате данное поведение будет выполнено только один раз.

2. Порядок выполнения работы

2.1. Написать код агента, создающего и посылающего сообщение в соответствии с приведенной ниже таблицей заданий (по вариантам).

Примечание. Данные, заносимые в поля агентов из столбцов «язык», «онтология» и «содержание» не имеют синтаксического значения и могут быть выбраны произвольно.

Таблица 1 - Варианты заданий

Вари-ант	Тип речевого акта	Агент - получатель	Язык	Онтология	Содержание
1	request	DA	English	Conversations	Stop talking
2	agree	AMS	KIF	Business	I'll do it
3	inform	Sniffer	Some-language	Weather	It's rain but very hot
4	request-when	RMA	My-language	My_ontology	Go! But sometimes.
5	query-if	AMS	Prolog	Parents	Son(John, Bob)
6	inform-if	Sniffer	JAVA	Relations	str != "abcde"
7	propose	DA	FIPA-SL	Some-ontology	Let's go home
8	inform-ref	RMA	Unknown	DistanceLearning	DistanceCourse-MAS
9	query	Sniffer	C++	Calculation	iCcounter >= 125
10	not-understood	DA	Human-language	Unknown	It doesn't make sense

Кроме указанных в таблице, необходимо также установить значения всех других слотов сообщения, выбрав их произвольным образом.

2.2. Откомпилировать класс агента, используя команду:

```
javac -<classpath> MyClass
```

здесь MyClass – имя класса пользовательского агента, код которого содержится в файле MyClass.java.

2.3. Запустить среду JADE, все необходимые инструменты и пользовательского агента. Включить подслушивание всех агентов (включая пользовательского) с помощью Sniffer. Просмотреть посланные сообщения средствами Sniffer или (когда возможно) - DummyAgent. Сохранить сообщения в текстовом файле средствами Sniffer.

2.4. Написать и откомпилировать второго пользовательского агента, отвечающего на сообщение, посылаемое первым агентом в соответствии с таблицей 3. (Варианты ответов придумать). Использовать при получении класс MessageTemplate. Перенаправить сообщение, посылаемое первым агентом от инструментов JADE второму пользовательскому агенту (изменить получателя сообщения). Запустить обоих агентов и инструменты JADE. Включить подслушивание агентов просмотреть и сохранить сообщения.

3. Содержание отчета

3.1. Цель работы.

3.2. Формат сообщений в языке FIPA ACL.

3.3. Исходный код пользовательских агентов на JAVA.

3.4. Текст сообщений, выведенных в консольном окне JAVA, и сохраненных средствами Sniffer.

3.5. Выводы.

4. Вопросы для самопроверки

4.1. Поясните назначение всех полей сообщения на языке FIPA ACL.

4.2. Перечислите и охарактеризуйте основные типы сообщений (речевых актов) в языке FIPA ACL.

4.3. Как используется класс ACLMessage? Перечислите основные методы этого класса.

4.4. Как используется класс MessageTemplate? Перечислите основные методы этого класса.

4.5. Описать возможные способы пересылки сообщений между агентами?