FIPA-OS TaskGenerator Tool

This tool can be used to auto-generate the message handling code associated with a FIPA protocol. The tool analyses a Java class file that contains a FIPA-OS protocol definition and auto-generates code that will handle the sending and receipt of messages associated with that protocol for both the initiator and the participant agents of a conversation.

To be precise, this tool will generate Java source files that define the five classes needed to handle both side of a conversation. They are:

The Initiator Handler Task

The initiator handler task contains methods for sending the messages that can be sent by the conversation initiator agent, and also methods for receiving messages that can be sent by the conversation participant agent.

The Initiator Ability

The initiator ability is an interface that must be implemented by the initiator agent. It contains method signatures for callback methods that are invoked on the agent when a message is received by the initiator handler task.

The Participant Handler Task

The participant handler task contains methods for receiving messages that are sent by the conversation initiator agent. It also contains methods for sending the messages that can be sent by the conversation participant agent.

The Participant Ability

The participant ability is an interface that must be implemented by the participant agent. It contains method signatures for callback methods that are invoked on the agent when a message is received by the participant handler task.

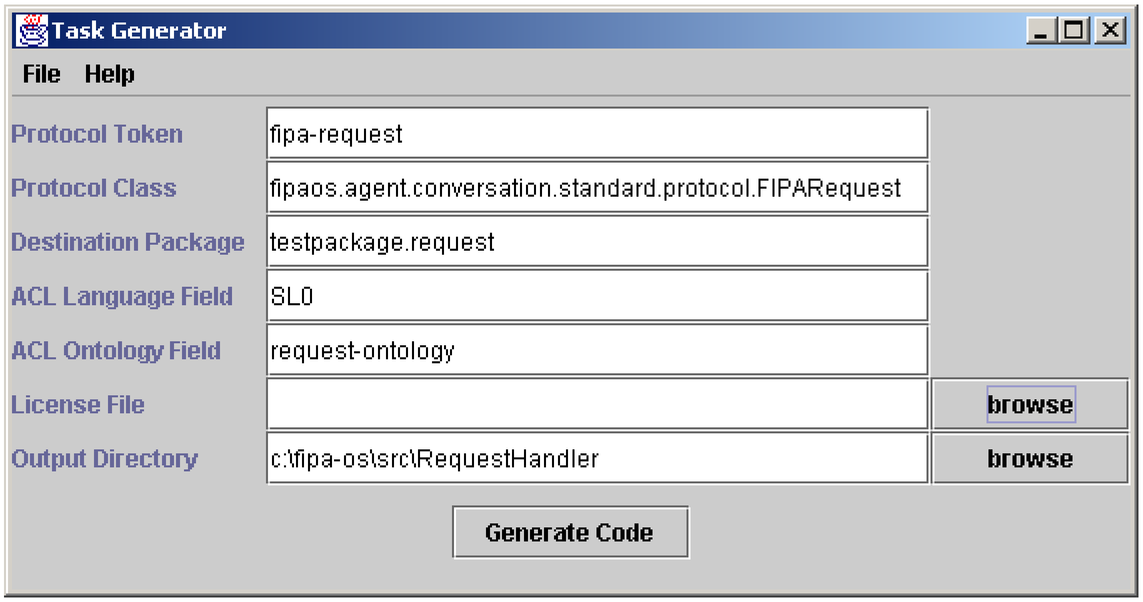
The Daemon Task

The task generation tool creates a daemon task that the participant agent sets as its listener task. This daemon task listens for the first performative in a conversation and when it receives that performative it spawns a new Participant Handler Task and passes it the conversation to deal with.

The generated class names are prefixed with the name of the class that defines the protocol. So for the FIPA request protocol, which is defined in a class called FIPARequest, the tool would generate the classes:

FIPARequestDaemonTask.java
FIPARequestInitiatorAbility.java
FIPARequestInitiatorHandlerTask.java
FIPARequestParticipantAbility.java
FIPARequestParticipantHandlerTask.java

Using the TaskGenerator Tool



Protocol Token	fipa-request
Protocol Class	fipaos.agent.conversation.standard.protocol.FIPARequest
Destination Package	testpackage.request
ACL Language Field	SL0
ACL Ontology Field	request-ontology
License File	
Output Directory	c:\fipa-os\src\RequestHandler

Using the tool is as simple as filling in the required fields and pressing the "Generate Code" button. The meaning of the fields is described below.

Protocol Token
The identifier for the protocol as it appears in the protocol field of a FIPA ACL message.

Protocol Class
The fully qualified name of the class containing the FIPA-OS definition for this protocol. The compiled class file that represents this protocol class must be in the Java classpath when the tool is run.

Destination Package
The Java package that you wish the generated classes to be placed in.

ACL Language Field
The value for the language parameter of ACL messages sent from the generated code.

ACL Ontology Field
The value for the ontology parameter of ACL messages sent from the generated code.

License File
If you wish, the tool can insert a license file or other header information as a Java comment at the start of each generated file. You can use the browse button to search for the license file using a file dialogue.

Output Directory
The location to which the generated Java files will be saved. You can use the browse button to specify the output directory using a file dialogue.

Once you have defined a protocol generation template (a *genmap*), you can save it to disk in case you need to regenerate the Java code again. The file menu allows you to load and save .genmap files as XML structures on disk.

Generated Code

The Handler Tasks

In the FIPA request protocol the initiator can send the performative *request*

The participant can send the performatives: *agree, refuse, inform, failure*

Therefore the FIPARequestInitiatorHandlerTask will contain the methods:

handleAgree
handleRefuse
handleInform
handleFailure
sendRequest

The FIPARequestParticipantHandlerTask will contain the methods:

sendAgree
sendRefuse
sendInform
sendFailure

Notice that the participant handler task does not contain a handleRequest method for the first performative sent by the initiator agent. This initial performative is handled by the participant handler task's startTask method. This is because the participant agent's daemon task has already intercepted the request message and passes it to a new FIPARequestParticipantHandlerTask through its constructor.

The Ability Interfaces

The interface methods are named in the format dealWith<protocol class name><performative to handle>

The FIPARequestInitiatorAbility interface will define the signatures:

dealWithFIPARequestAgree
dealWithFIPARequestRefuse
dealWithFIPARequestInform
dealWithFIPARequestFailure

The FIPARequestParticipantAbility interface will define the signatures:

dealWithFIPARequestRequest

Defining a FIPA-OS protocol

With the introduction of the TaskGenerator tool, the format of a FIPA-OS protocol definition has changed slightly. The agent action component is now mandatory instead of optional.

The format for a stage in a protocol is now:

```
(<performative> <agent action> <sender> [next stage])+
```

As an example, the code for the FIPARequest protocol is given below:

```
private static Object[] __after_agree =
{
FIPACONSTANTS.FAILURE, new Integer(CONVERSATION_END), new Integer(1),
FIPACONSTANTS.INFORM, new Integer(CONVERSATION_END), new Integer(1)
};

private static Object[] __after_request =
{
FIPACONSTANTS.NOT_UNDERSTOOD, new Integer(CONVERSATION_END), new Integer(1),
FIPACONSTANTS.REFUSE, new Integer(CONVERSATION_END), new Integer(1),
FIPACONSTANTS.AGREE, new Integer(AGENT_ACTION_REQ), new Integer(1), __after_agree
};

public static Object[] __protocol =
{
FIPACONSTANTS.REQUEST, new Integer(AGENT_ACTION_REQ), new Integer(0), __after_request
};
```