



AgentService Visual Studio Integrator

Create your agents with a click

The AgentService Team

Copyright @ 2009 – l.i.d.o. – DIST University of Genoa

Abstract

Here we present a component of the AgentService framework aimed to the easy design and implementation of *agents*, *behaviours* and *knowledge objects*.

This brief guide shows how to install the *Visual Studio Integrator* and to use the editor in order to automatically generate agent templates.

Index of Contents

AgentService Visual Studio Integrator	1
Create your agents with a click	1
The AgentService Team.....	1
Copyright @ 2009 - i.i.d.o. - DIST University of Genoa	1
Abstract	2
Index of Contents.....	3
1. Introduction	4
2. Installation of the tool	4
3. Create a multi-agent application.....	5
3.1. The new solution	7
3.2. Designing an agent	8
3.3. Towards a complex diagram	11
3.4. Importing agent templates.....	12
3.5. Let's fill the classes.....	13

1. Introduction

The *AgentService Visual Studio Integrator* is a tool which enables the designer to easily design and implement a multi-agent platform, by using *Visual Studio* in connection with *Microsoft Office Visio*.

The tool supports both *Visual Studio 2005* and *2008* and requires the previous installation of *Visio 2003* or *2007*. By creating a new solution for an AgentService platform, the user will be able to compose a diagram describing *agents* with their *behaviours* and *knowledge objects*, and automatically the tool will generate the related C# classes interconnected on purpose.

2. Installation of the tool

The installation process is very simple. Before to install the tool, check if in your PC are installed:

- *Office Visio 2003* or *Office Visio 2007*
- *Visual Studio .NET 2005* or *Visual Studio .NET 2008*

Then, unpack the zip file containing the *Visual Studio Integrator* and run *VSVisioEditor Setup.msi*.

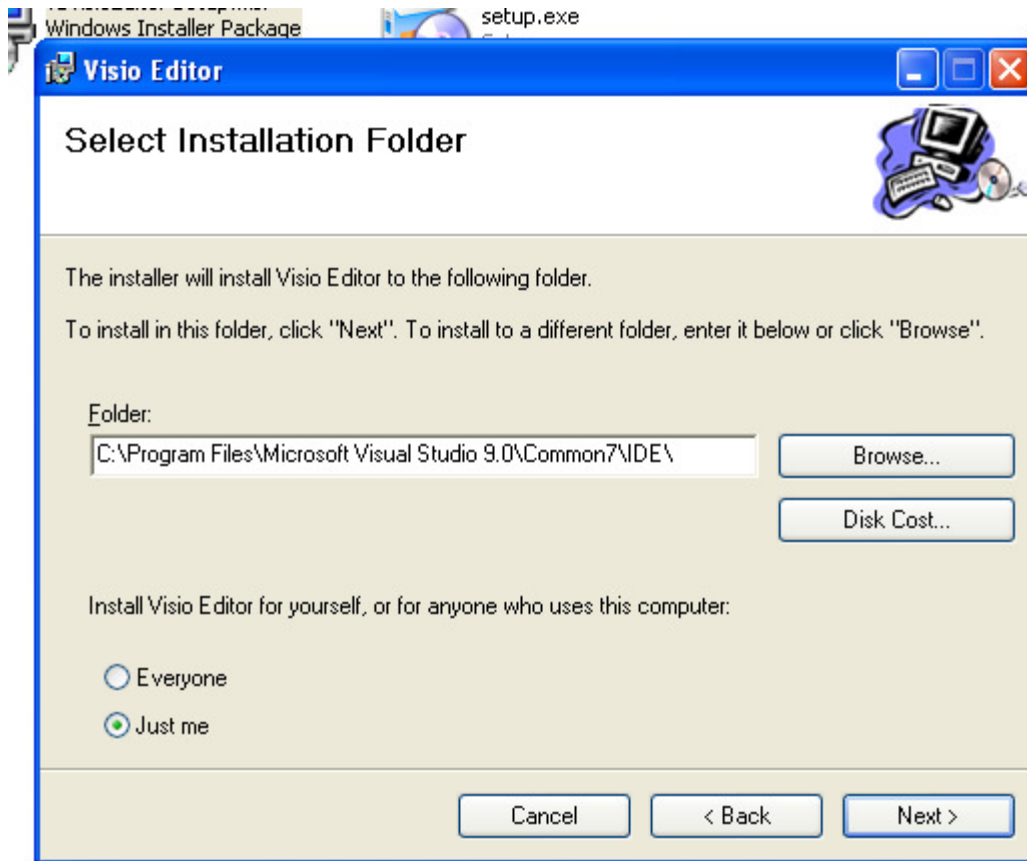


Figure 1: choose the installation directory.

As shown in Figure 1 you have only to check if the installation wizard has found the correct folder for your local Visual Studio deployment.

Wait until the installation process terminates.

3. Create a multi-agent application

Now you are able to create an AgentService platform by using Visual Studio in connection with the Visio Editor. Open Visual Studio, go to *File>New>Project*. As shown in Figure 2, in *Project types*, select *Visual C#* and in *Templates*, under *My Templates*, select the project template *AgentServicePlatform*.

Type the name of you AgentService application and click OK.

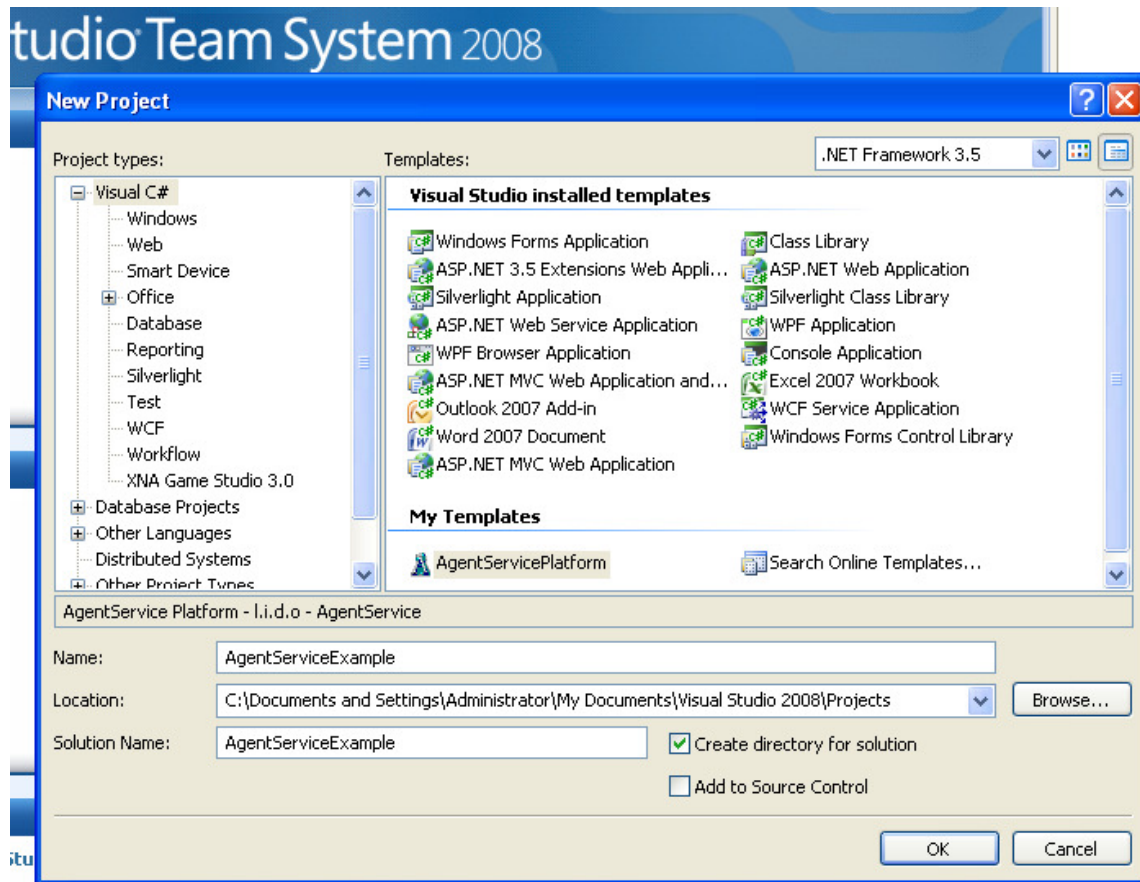


Figure 2: create a new project.

3.1. The new solution

Visual Studio opens a new solution and if you watch the *Solution Explorer*, you can notice some things:

- *Two references present a warning*: you have to reload these two references with your local AgentService assemblies.
- *Batch.cs*: it contains the batch class to run your application. As you see later, you have only to add few code lines to this file.
- *AgentServicePlatformFile.vsd*: it is the Visio diagram which will contain the graphical representation of your platform.

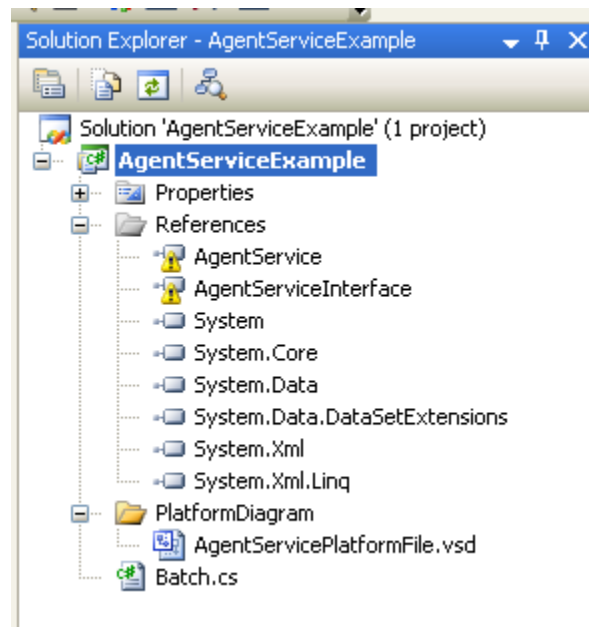


Figure 3: the solution explorer.

Click on the *AgentServicePlatformFile.vsd* and a new page will appear, with an editor identical to the Visio one.

Figure 4 shows the editor and the relative toolbox which contains the shapes that can be included in the diagram.

- *Agent*: an agent icon
- *Behaviour*: the behaviour icon which can be connected to agent icons.
- *Knowledge*: the knowledge icon which can be connected to the behaviour icons.
- *Connector*: the segment which connects two shapes.
- *Import DLL*: it allows the possibility to import whole agents or only behaviours or knowledge objects from selected assemblies.

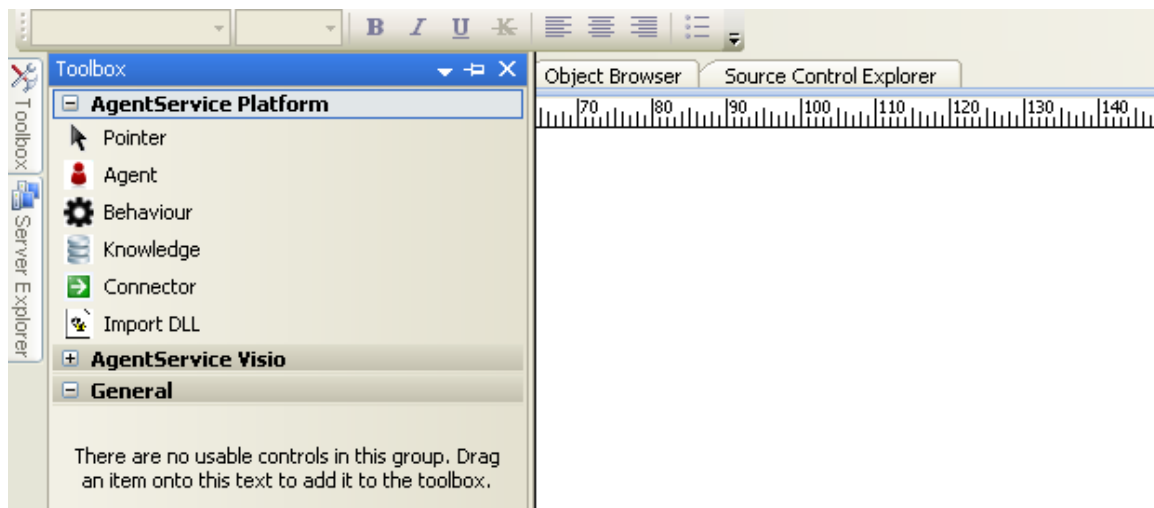


Figure 4: the AgentService toolbox.

3.2. Designing an agent

From the aforementioned toolbox, you can get the elements which will compose your platform diagram.

Click on the *agent* icon (drag and drop is currently not available). A configuration window appears and you must insert the *name* of the agent template, which is also the name of the class representing the agent (see Figure 5).

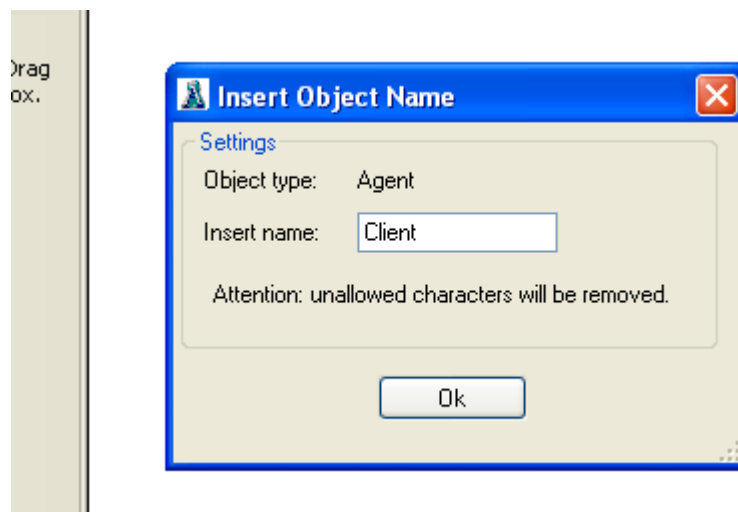


Figure 5: the agent configuration window.

Click *Ok* and a new icon appears in the Visio diagram and contemporary, a new class with the name of the agent template is created in the solution. In this case the class name is *Client* (Figure 6).

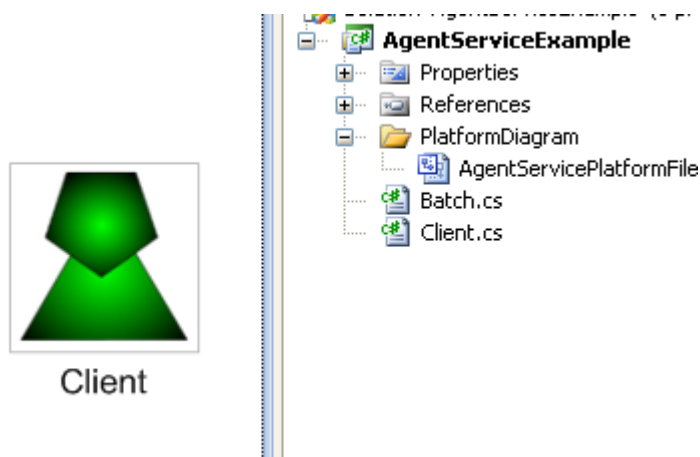


Figure 6: a new agent template.

Visual Studio automatically opens the *Client.cs* file and you can see that the skeleton of the class is completely generated in an automatic way.

Now click on the *behaviour* icon and set the *name* of a new behaviour in the window which will appear. A new icon is created on the diagram and a new class is included in the solution. In the same way, click on the *connector* icon and link the *agent* icon with the *behaviour* one (Figure 7).

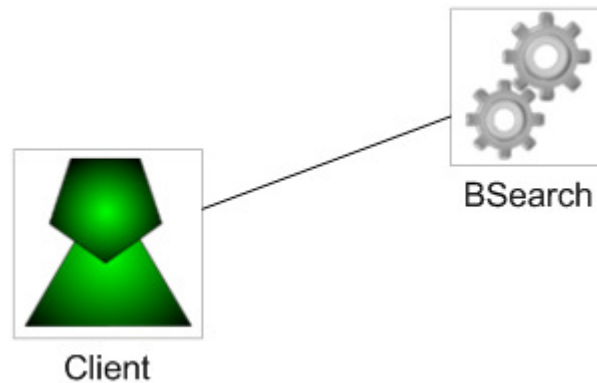


Figure 7: an agent with its behaviour.

Open the *Client.cs* file and note that before the class definition a new code fragment has been automatically created:

```
[AgentBehaviours(typeof(BSearch))]  
[Serializable()]  
public class Client : AgentTemplate  
{
```

In this way the new behaviour is linked to the agent template. Try to add a knowledge object to the behaviour. Click on *knowledge* icon, set a proper *name* (in this example *KBasicInformation*) and link the knowledge object to the behaviour, as shown in Figure 8.

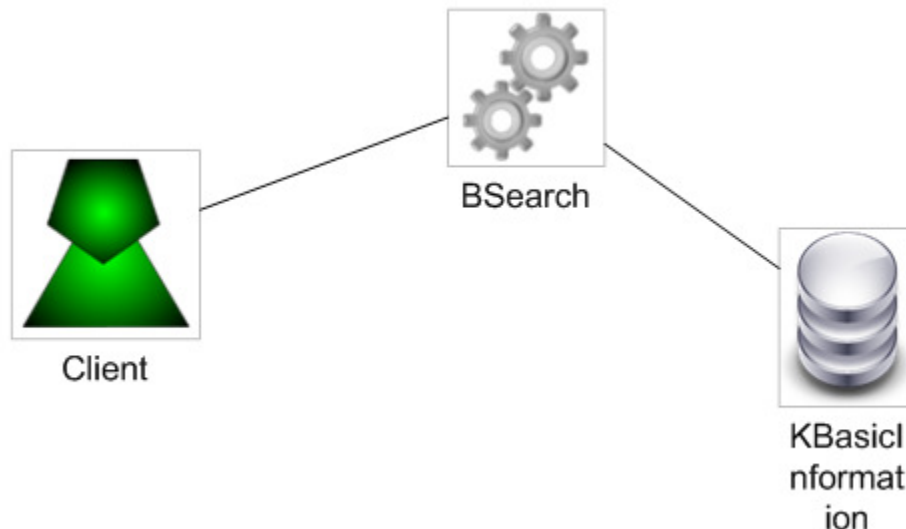


Figure 8: linking a knowledge object.

The following code fragments shows that both the *Client* and *BSearch* classes are updated.

Client.cs:

```
[AgentKnowledges(typeof(KBasicInformation))]
[AgentBehaviours(typeof(BSearch))]
[Serializable()]
public class Client : AgentTemplate
{
```

BSearch.cs:

```
[Knowledge(typeof(KBasicInformation))]
public class BSearch : Behaviour
{
```

In the same way you can remove shapes from the diagram and the related classes are consequently updated.

3.3. Towards a complex diagram

You can add more agent templates, creating a potentially complex diagram.

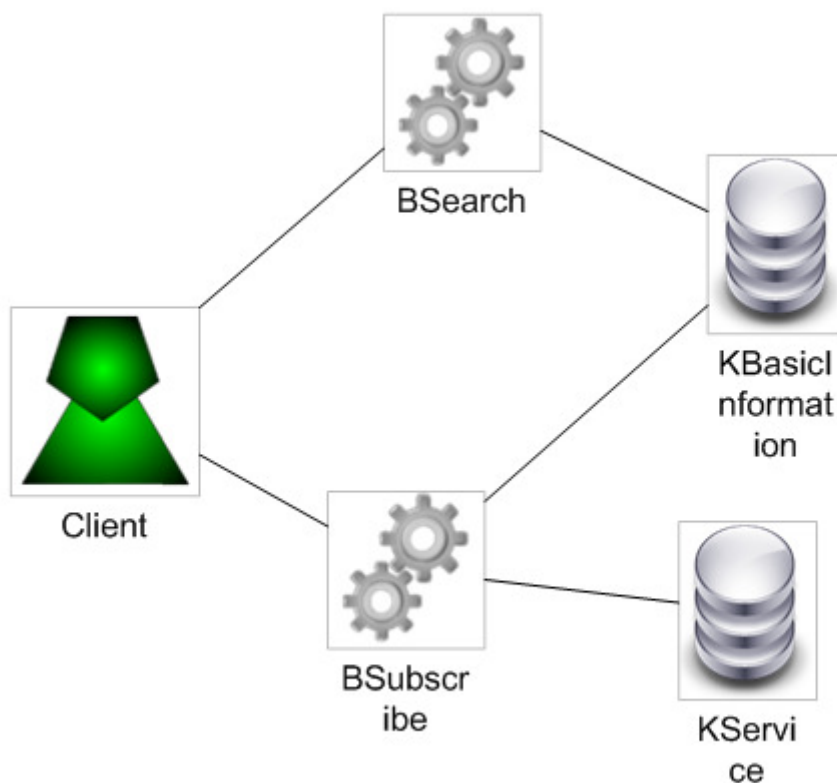


Figure 9: a new behaviour with two knowledge objects.

Figure 9 shows a behaviour called *BSubscribe* which has two knowledge objects: *KService* and *KBasicInformation*. In this case *KBasicInformation* is shared

between *BSubscribe* and *BSearch*. This means that the *KBasicInformation* instance is the same for both the behaviours.

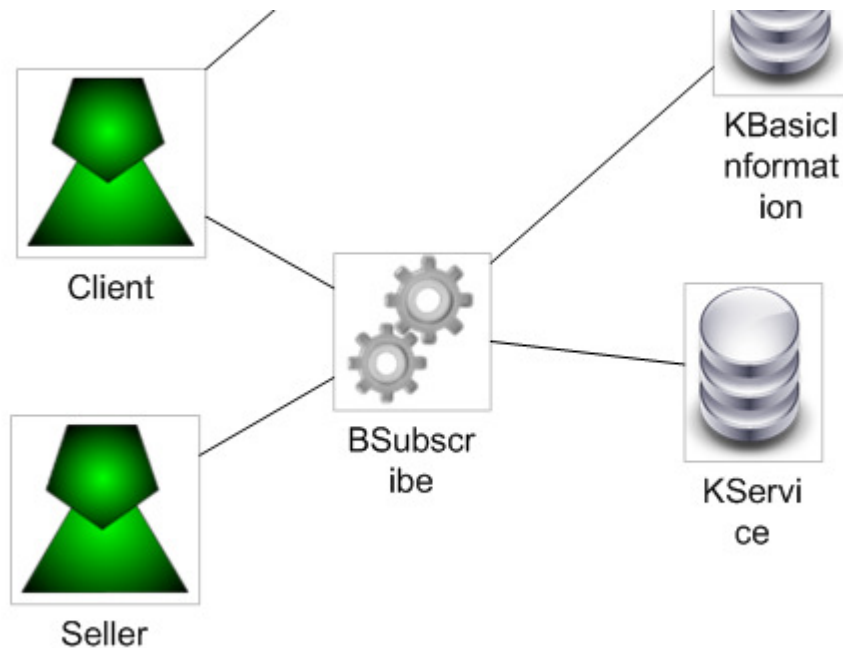


Figure 10: a new agent.

In Figure 10 the new agent template *Seller* uses the same behaviour of the *Client* agent. In this case, every agent template shared only the definition of *BSubscribe* and then of *KService*, but their instances are different. So every agent instance has its own *BSubscribe* thread and its own knowledge object with different data which are not shared among agent instances.

3.4. Importing agent templates

You can also import definitions of agents directly from assemblies. In this way, you can reuse agent definitions, importing the whole *agent* (included *knowledge objects* and *behaviours*), only *behaviours* and *knowledge objects*, or only *knowledge objects*.

Click on the icon *Import DLL* in the toolbox. In the new window, click on *Load*, select a DLL containing agent templates, and click on *Analyze* (Figure 11).

In the *Details* section are shown the information about all the entities which can be imported. You can select them in order to include them in your diagram.

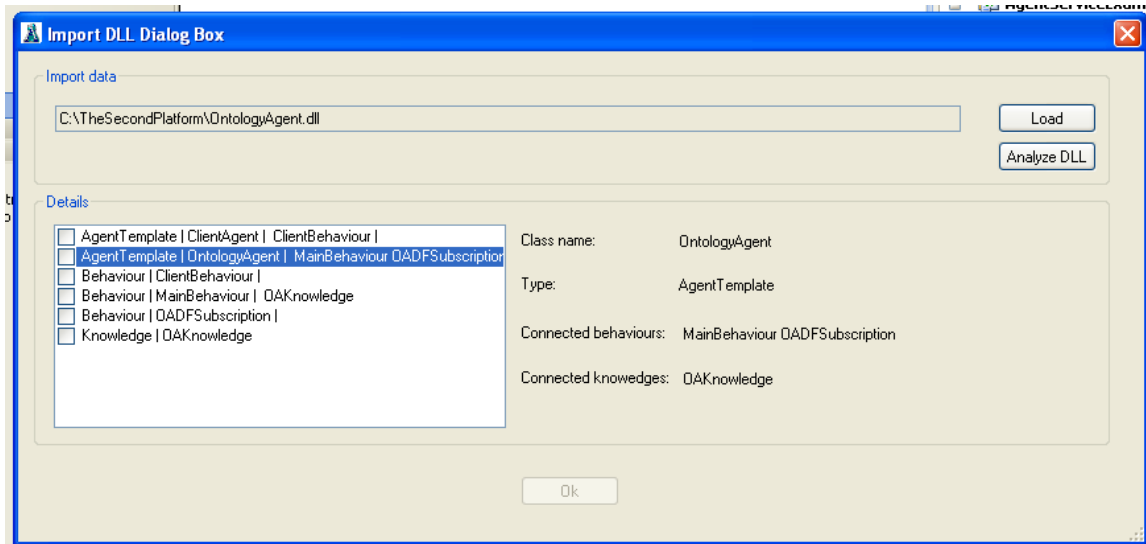


Figure 11: import a DLL.

In this case we have imported from the *OntologyAgent.dll* assembly the *OntologyAgent* template along with its *MainBehaviour* and the *OAKnowledge*. In the *references* list appears the aforementioned assembly: *OntologyAgent.dll*.

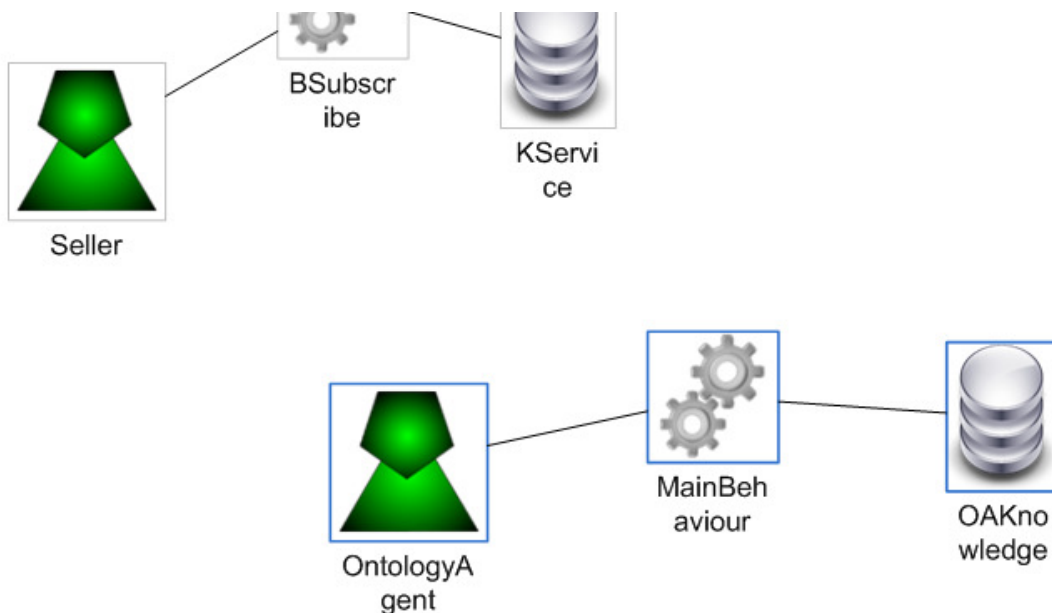


Figure 12: an imported agent.

3.5. Let's fill the classes

Now you are able to complete the classes with your business code. Remember to manage the three basic methods of your agents: *Initiate*, *Terminate*, and *HandleConversation*. For your behaviours add your code in the *Body* method.

Regarding the knowledge objects, remember to complete the input parameters of the *constructor*, in order to pass the requested objects to initialize the knowledge instance.

Finally, in the *batch* class, before *controller.Services.ActivateAllAgents()* add the directives for instantiating your agents.