
# OPEN Project

## STREP Project FP7-ICT-2007-1 N.216552

| | |
|---|---|
| **Title of Document:** | Prototype for Application Logic Reconfiguration |
| **Editor(s):** | H. Klus, B. Schindler, C. Deiters |
| **Affiliation(s):** | Clausthal University of Technology |
| **Contributor(s):** | |
| **Affiliation(s):** | |
| **Date of Document:** | 31.03.2009 |
| **OPEN Document:** | WP 4, D4.3 |
| **Distribution:** | Public |
| **Keyword List:** | Adaptive Systems, System Reconfiguration, Migration, OSGi |
| **Version:** | Final |

**OPEN Partners:**

CNR-ISTI (Italy)
Aalborg University (Denmark)
Arcadia Design (Italy)
NEC (United Kingdom)
SAP AG (Germany)
Vodafone Omnitel NV (Italy)
Clausthal University (Germany)

| **Title:** Prototype for Application Logic Reconfiguration | **Id Number:** WP 4, D4.3 |
|---|---|

# Abstract

This document describes a prototype which illustrates techniques of application logic reconfiguration. This prototype has already presented at the last review meeting in March 2009.

The prototype is related to deliverable D4.1 (Solutions for Application Logic Reconfiguration) as it shows how solutions for application logic reconfiguration can be applied. As example application we will use the well-known PacMan game, a simple arcade game. The application will show how the PacMan game is adapted during its migration from a PC to a so called mini-PC.

# Table of Contents

# 1 Application Logic Reconfiguration Scenarios within the Prototype

In this section we will introduce the prototype in more detail. To do this, we will first introduce the PacMan game itself, and afterwards the several migration and adaptation steps.

## 1.1 The Original PacMan Game

PacMan is a game where a character called PacMan, which is steered by the user, has to collect dots in a maze, like shown in Figure 1.

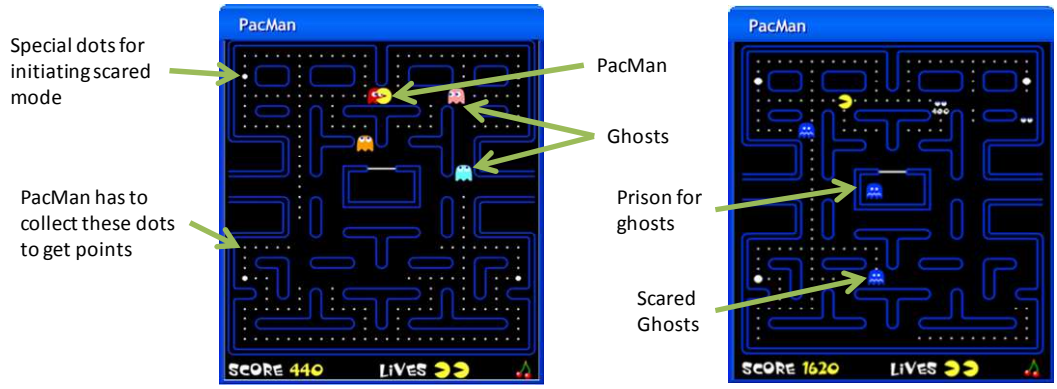


**Figure 1:** The two modes of a single PacMan game. On the left hand side the game is in normal mode where the ghosts try to catch the PacMan. On the right side the game is in scared mode where the PacMan can catch the ghosts.

Ghosts, who are controlled by the computer, are running around with the goal to catch the PacMan. If the PacMan collects special dots, ghosts and PacMan change roles for some seconds like depicted in Figure 1 on the right hand side. That means that the PacMan now can catch ghosts and that the ghosts try to run away. Caught ghosts will be imprisoned in the middle of the maze for some seconds. After some seconds the roles change back again. The goal for the player is to get as much scores as possible by collecting dots and catching ghosts.

## 1.2 Initial Situation

After starting the game on the PC, it will look like depicted in Figure 2. Five ghosts try to catch the PacMan while the PacMan tries to collect all dots in the maze. The PacMan will get one point for collecting one dot. At the bottom of Figure 2, the component is depicted which implements the game functionality. It has two configurations like introduced in deliverable D4.1. As no *AccelerationSensorIf* is available, the component runs in configuration as indicated by the green check.

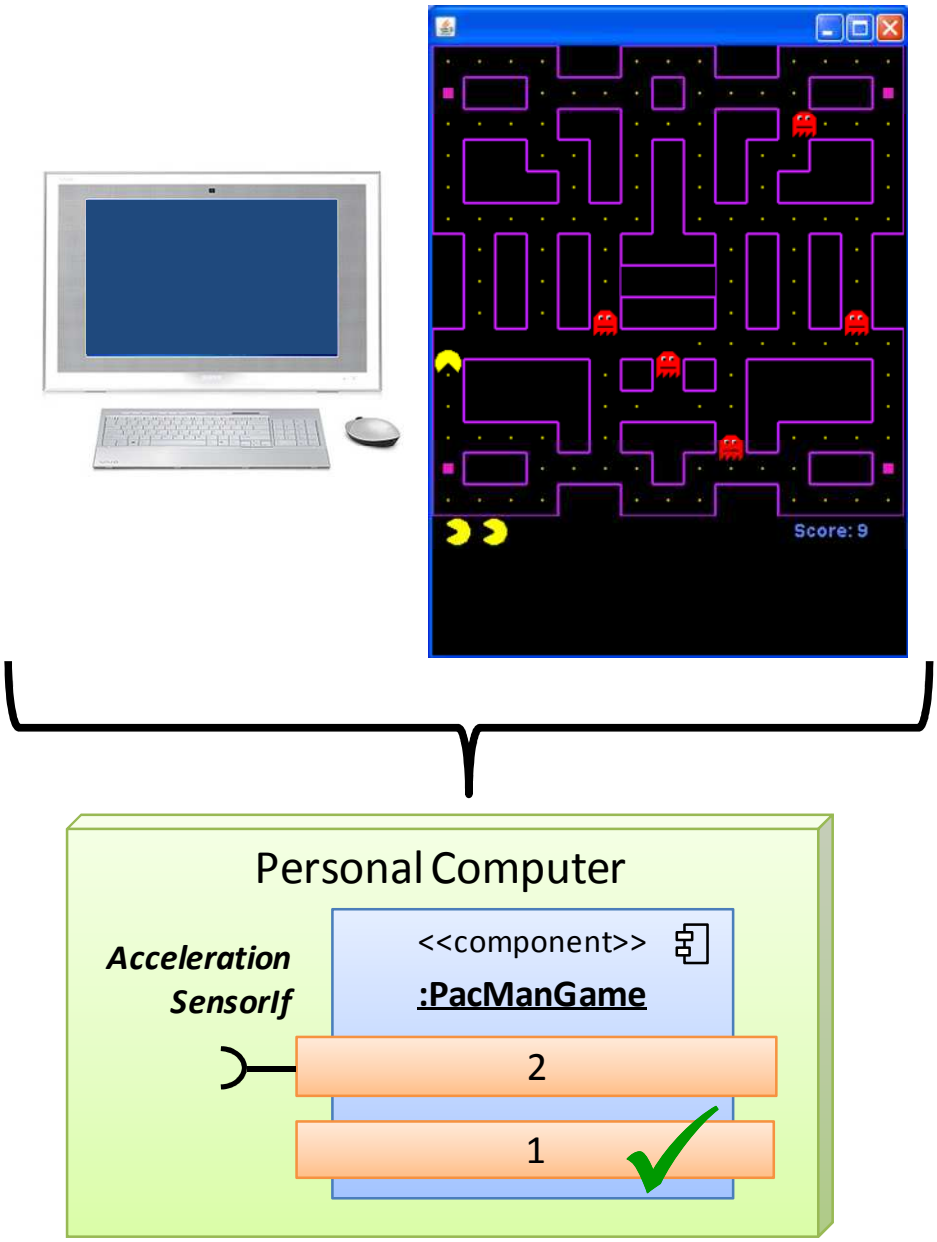


**Figure 2:** The initial situation after starting the PacMan game on a personal computer. On the bottom you see the component which realizes the game.

The *PacManGame* component itself consists of several subcomponents distinguished among others into user interface subcomponents and application logic subcomponents.

## 1.3 Service Usage Adaptation

We will now mainly show how service usage adaptation, introduced in deliverable D4.1, takes place in this prototype.

The user switches on an accelerometer and connects it to the personal computer. To do this, we use in the prototype a so called Sun Spot. It is programmable with

Java and enables easy access to sensor data like acceleration. If the user now tilts the Sun Spot, the PacMan will move into the according direction. Figure 3 shows the new application configuration.

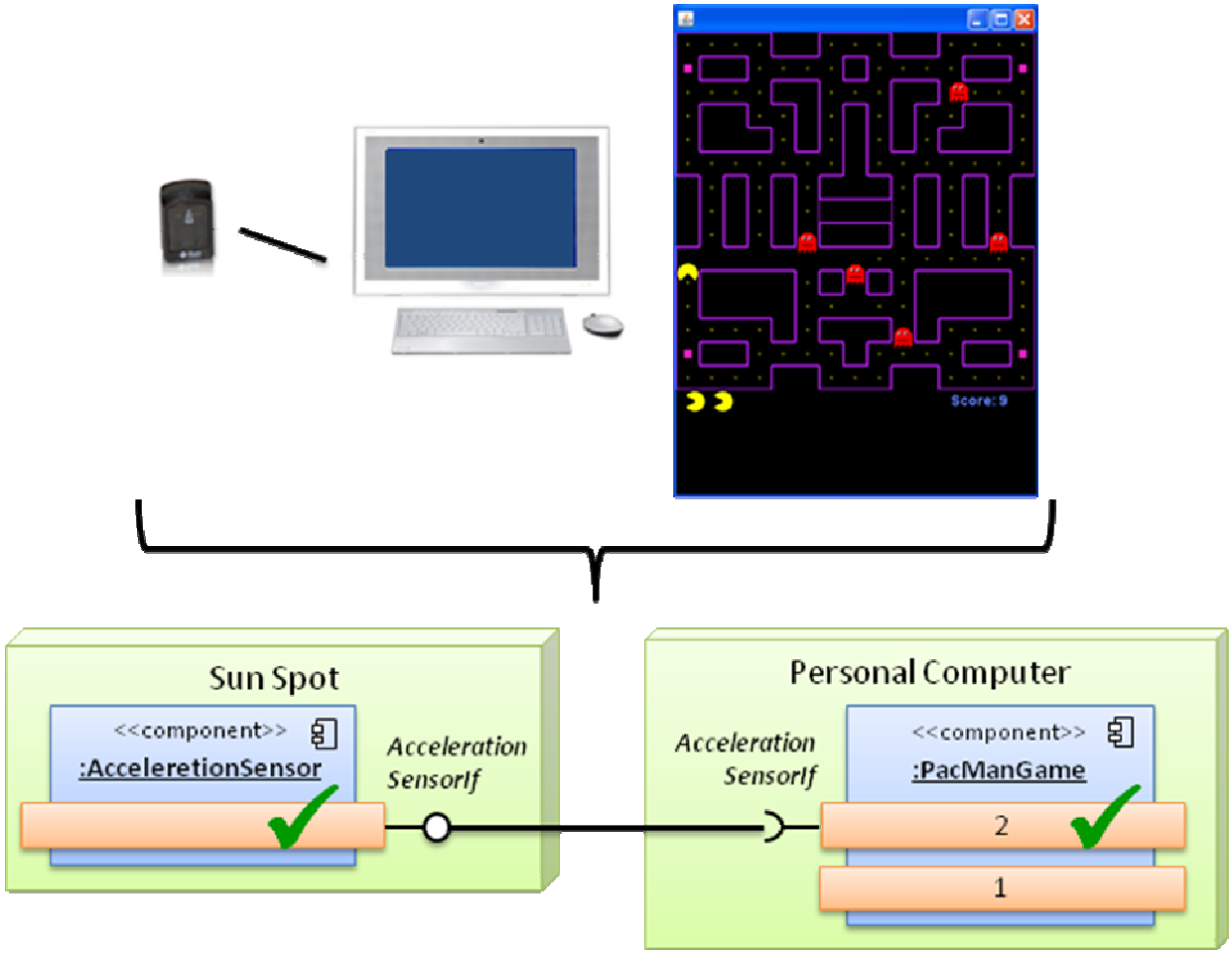


**Figure 3:** Situation after integrating an accelerometer into the game during runtime.

The *PacManGame* component now retrieves the moving directions of the PacMan via the *AccelerationSensorIf*. Therefore, the configuration of the *PacManGame* component is switched to configuration 2, indicated by the green check. In the same time, the application logic is adapted. As steering is now more difficult as before, the speed of the ghosts is slowed down. To summarize: Configuration 1 takes input from the keyboard and sets speed of ghosts to average, Configuration 2 takes input from the acceleration sensor and sets speed of ghosts to slow.

## 1.4 Service Behaviour Adaptation

If the user now wants to continue playing the game on a mobile device like a mini-PC for example, a middleware user interface can be used to select the target and what to migrate like shown in Figure 4.

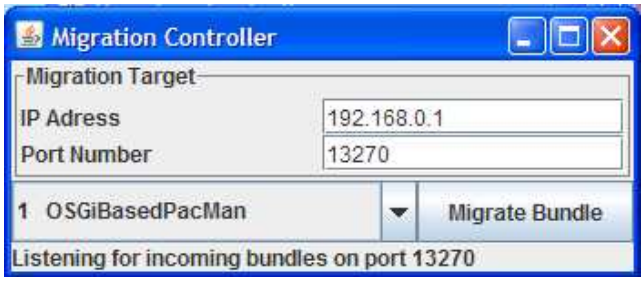


**Figure 4:** The migration controller enabling the user to chose the target device, the application to migrate, and to execute migration.

Here the user can first define the target device by entering the IP address and the according port number. Using a drop-down menu, the user can then choose what to migrate. In this case it is the PacMan game. By clicking on *Migrate Bundle* the middleware will transfer the code of the PacMan game to the target device. The target device runs also a Migration Controller which receives the code, and restarts it and adapts its behaviour to the device like introduced in the following.

Figure 5 shows the game after the migration from the PC to a mini-PC. The main differences between the mini-PC and a PC are its display size and its options to steer the PacMan. Using the small keys of the mini-PC keyboard makes it more difficult to steer the PacMan than using a standard keyboard. And the small display size makes it difficult to see the dots in the maze if using the PC version of the PacMan game. Thus, the game has to be adapted in two ways: Considering the small screen and the more difficult steering of the PacMan.

The overcome the disadvantages of the small screen, first the size of the dots is increased and at the same time the number of dots is decreased. This is mainly user interface adaptation. But the application logic has also to be adapted. As now fewer dots are available, collecting one dot must now result in giving a higher score. In this prototype the number of dots is halved, and therefore collecting one dot will result in getting two points for collecting one dot instead of one point.

To cope with the more difficult steering, the speed of the ghosts is adapted, as already shown before. At the bottom of Figure 5, the current configuration of the *PacManGame* component is shown.

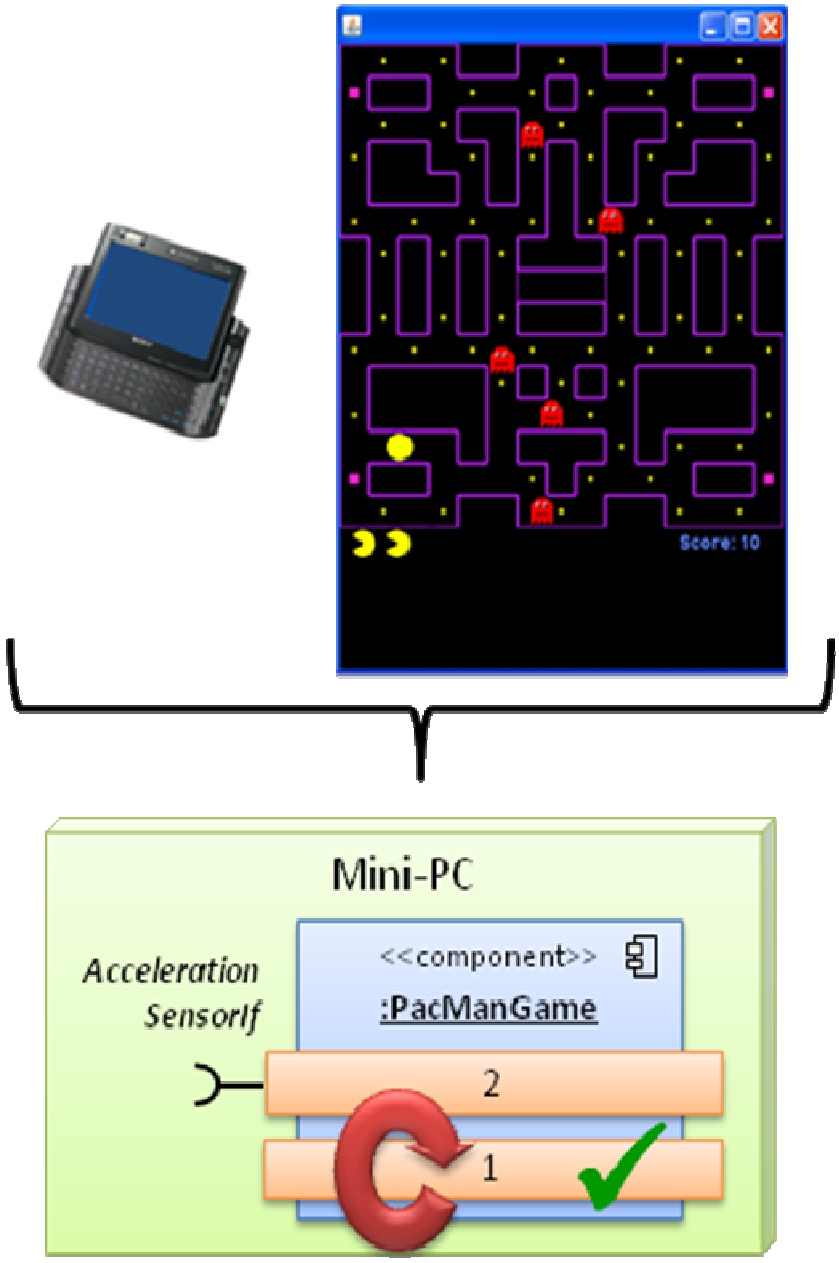


**Figure 5:** Situation after migration of the game from the PC to a mini-PC.

As the *AccelerationSensorIf* is not available, the component runs in configuration 1. But the internal realization of that configuration is adapted according to context information like display size for example. In deliverable D4.1, this kind of adaptation has been introduced as *Service Behaviour Adaptation*.

# 2 Runtime Environment and Installation

We will now outline the programming and runtime environment and give some installation instructions.

The whole application has been realized using OSGi. OSGi is a middleware enabling the development of component-based Java applications. Thus, the game as well as the middleware is implemented in Java. In OSGi components are called *bundles*. In this application there are on the one hand the bundles realizing the PacMan game. On the other hand, the *Migration Controller* is also implemented as an OSGi bundle. We chose OSGi for several reasons:

- The OSGi runtime environment needs only a few kilobytes of memory which makes it possible to run applications even on embedded devices.
- The development of bundles is easy and not burdened with too much middleware aspects. In fact, standard Java code can be converted into a bundle quit easily. Furthermore, the Eclipse IDE offers the functionality to run bundles directly without the need to install them first in an OSGi framework.
- There exist several open source OSGi framework implementations for different platforms and devices like Knopflerfish, Concierge or Oscar.
- OSGi offers already some functionality for the integration of new components during runtime, as well as the possibility to migrate code from one device to another.

Thus, the prototype comes as a set of OSGi bundles which can be installed and executed in arbitrary OSGi runtime environments. Such OSGi bundles come as JAR-files. The Migration Controller for example comes as MigrationControl.jar.

To run the bundles, start first an OSGi framework implementation. We tested among others a framework called *Knopflerfish* (http://www.knopflerfish.org/index.html). To load the JAR-files into the framework, you can use the menu of Knopflerfish as shown in Figure 6.

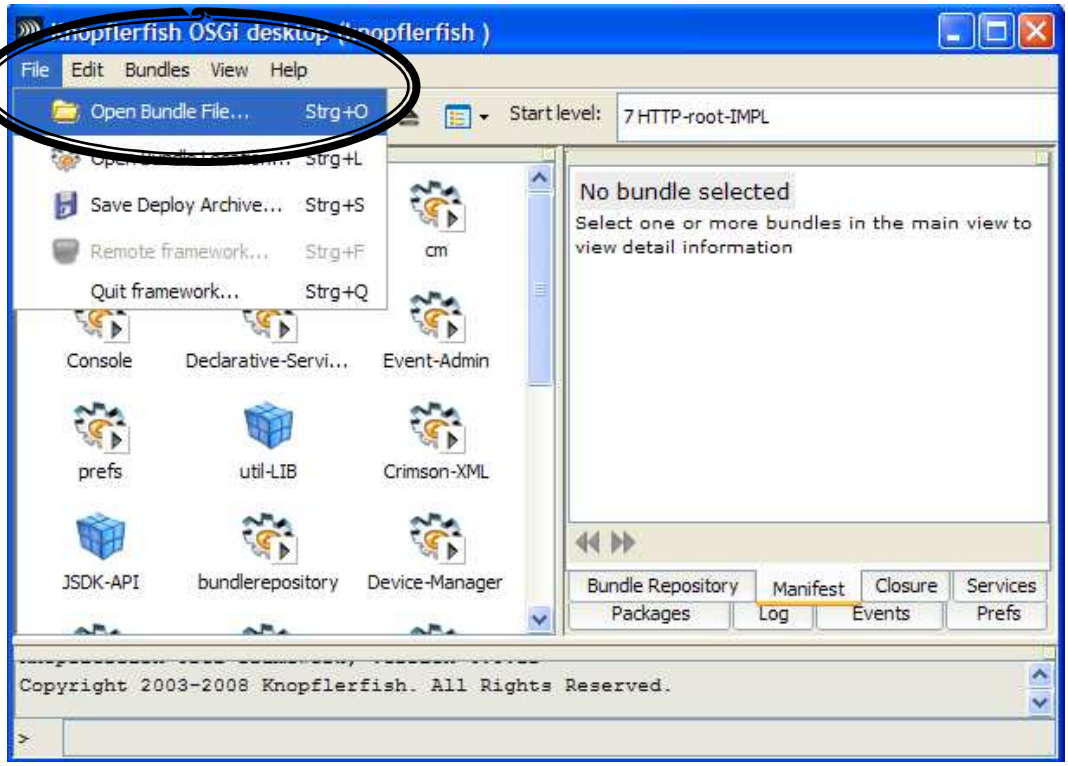


**Figure 6:** Knopflerfish menu to load a bundle into the framework.

After the bundles have been loaded into the framework, they can be executed. Figure 7 shows the PacMan game loaded into the OSGi framework. By marking that bundle and clicking on the play button on the top, the game will be executed and the game window will appear. You can use the menu also to stop and restart the game, and also to uninstall the bundle from the framework.

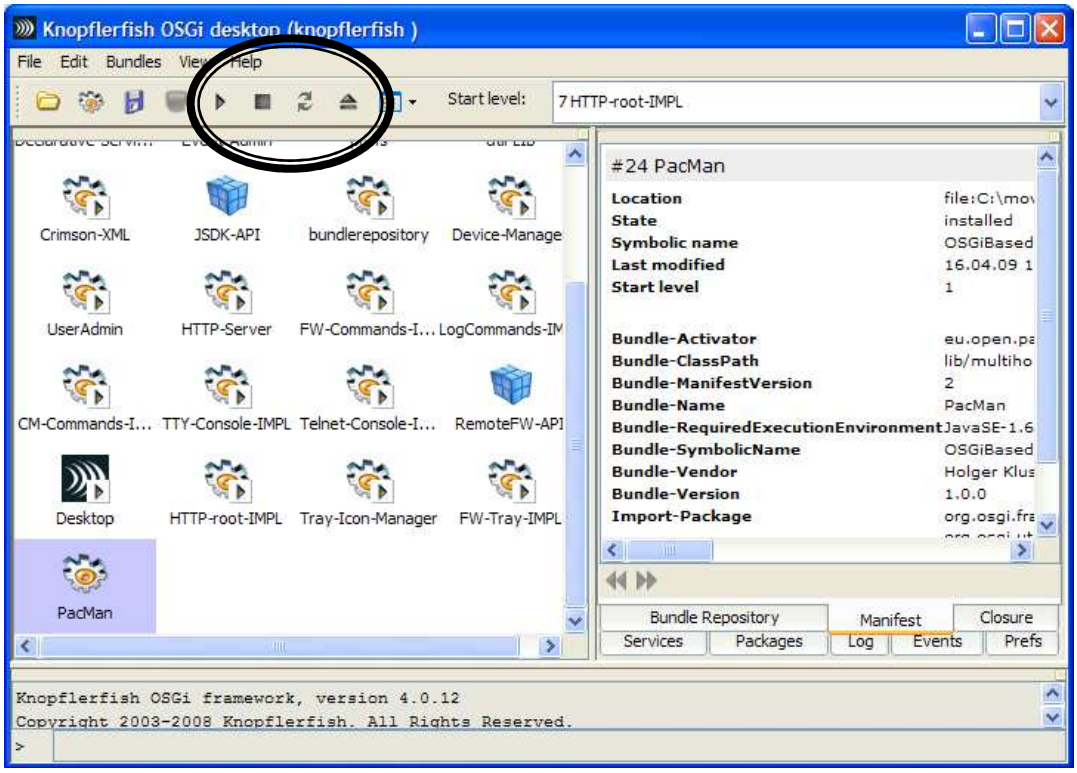


**Figure 7:** The PacMan game installed as an OSGi bundle into an OSGi framework called *Knopflerfish*.

On the right hand side of Figure 7 you can see additional information about the marked bundle like the version and vendor. The *Migration Controller* can be installed and executed in the same way. After starting the Migration Controller bundle, the window depicted in Figure 4 will appear. Afterwards, the GUI of the Migration Controller can be used for migration of arbitrary bundles from one device to another as explained in the section before.