Abstract

This paper deals with the question how to integrate smart devices in Java applications. It will outline how different smart devices can be used to enrich learning environments, we will point to some of the problems one has to face while dealing with smart devices, a differentiation of smart devices will be done and we will give an overview about existing Java Virtual Machines available for different smart devices. Furthermore we will tackle the question of the communication between different smart devices and also between different kinds of smart devices. An outlook to the future work will also be given at the end of this work.

1 Introduction

Nowadays smart devices like cell-phones or PDA’s (Personal Digital Assistants) belong to our every day live. The usage of those devices has become more and more natural within the last few years. Furthermore there exist devices like programmable Lego bricks, scanners that are not much bigger than a pen or digital cameras. Usually those devices are not used in an interoperable way but more or less stand-alone. Given that smart devices can enrich learning environments they can even more enrich the learning environment if they are used in combination. Being aware that the interoperable usage of these devices is not always an advantage (and even more not always possible), we try to implement a framework that supports the integration and interoperability in a flexible way.

There are a lot of reasons why some smart devices may not interact with others, e.g. combining a digital camera with a programmable Lego brick does on the one hand make no sense because the calculation power and the space for programs on the Lego brick will not allow image processing and on the other hand the interfaces of these two devices will usually be incompatible.

Another example that would make sense is the usage of a mobile phone in connection with a digital camera. Nowadays new models of cell-phones usually ship with their own digital camera to make images and send them via MMS. Assuming to be able to connect a normal digital camera, e.g. via IRDA, to a cell-phone it would not be necessary to have different digital cameras for different scenarios.
This paper will first show what kind of smart devices we want to integrate, afterwards we will give an overview about different Java Virtual Machines for smart devices, the first version of the framework will be presented and the question about the communication between different smart devices will be tackled. Last but not least we will present an outlook for future work and we will present open questions and problems with this framework.

2 Smart Devices

There are plenty types of smart devices available on the market. For example there exist little scanners, not bigger than a pen, cell-phones, digital cameras, PDA’s and even programmable Lego bricks. Since the variety of devices is quite huge it becomes more and more obvious that interaction between different smart devices is not an easy task. Also the interaction just with one smart device differs because of the different nature of smart devices. Most likely one will not assume a digital camera to be programmable or to exchange its data with a database, but for another smart device like a PDA it might be very useful to interchange data like calendar entries with a database.

This leads to a differentiation of smart devices by the way we can control the device itself and the data produced and consumed by the device. A first, very rough, differentiation might be to divide the device by the possibility to program it or just interchange data in any sense. By this a PDA is an example of a device we can surely program and a digital camera is a device we can just exchange data with by downloading the images that were taken by the camera.

A differentiation like this just gives a rough approximation about all smart devices. For example the degree of data exchange can still vary from data exchange in both way, so we will be able to upload data to the device and also download data from it, or data exchange just one way, to the device or from the device. One example for the data exchange in both ways might be a cell-phone that allows us to download the entries of a phonebook, edit them and update the changed entries later on to the phone.

A framework for integration of smart devices should handle those differences in a most transparent way by presenting interfaces and abstract classes that allow the interaction with different kinds of devices in the same way. In this paper we’ll present our approach build such a framework.

3 Learning Scenarios for Smart Devices

Smart devices can enrich the learning environment with a lot of new scenarios like shown in [1]. The combination of different smart devices could help to increase the learning value[2]. Using a PDA in combination with a programmable Lego brick creates opportunities for a scenario where students program physical robots that are able to find a way out of a maze.

Another advantage of smart devices is that they are usually portable. That a lot of new scenarios like easy to set up ad-hoc classrooms, data mining on a field trip or to integrate
the students much deeper into the problem by letting them be part of the problem that should be solved. Furthermore smart devices could easily be integrated in ubiquitous computing frameworks[3].

3.1 Learn smart programming with smart devices

For example could a Lego robot be put in a maze built of physical bricks and the robot tries to find his way out of the maze by following simple rules that are defined by conditions. If the robot doesn’t find a rule for a specific situation he can wait at his position for a student to come, the student can get a visualization of the situation and can define a rule on the PDA how the robot should react and upload the rule back to the robot that now can follow up in finding his way out of the maze. Additionally the students can formulate new rulesets by using a virtual maze with a simulated robot on the PDA, share those rulesets with other students by uploading them to a centralized server, or try to build up mazes that a robot can not escape from using a special ruleset.

3.2 Incorporate students or teachers outside the face to face scenario

Another possible scenario will be that a teacher uses a collaborative discussion tool allowing the students to send short messages from their cell-phones to the discussion space. A scenario like this would also allow students or teachers that are currently not available in the face to face scenario to participate in the discussion.

4 Java Versions for Smart Devices

There exist plenty of Java implementations on the market for smart devices. They are mainly for PDA’s but also versions for a programmable Lego brick are available. For cell-phones the thing is a bit different, most cell-phones that have Java support directly ship with a J2ME implementation.

4.1 Java 2 Micro Edition (J2ME)

Sun presents a Java solution for smart devices called J2ME\(^1\) (Java 2 Micro Edition) which mainly comes with two different configurations, the CDC\(^2\) (Connected Device Configuration) and the CLDC\(^3\) (Connected Limited Device Configuration). Sun goes for a differentiation for smart devices that differs from ours described in the previous section: they divide the device by the amount of memory and the computational power. For devices with less memory and less computational power they got the CLDC which lags for example the support for AWT and Swing. It neither supports RMI. This configuration is mostly found on cell-phones.

\(^1\)http://java.sun.com/j2me
\(^2\)http://java.sun.com/products/cdc
\(^3\)http://jcp.org/aboutJava/communityprocess/final/jsr139/index.html
For devices with more memory and more computational power, Sun developed the CDC which supports AWT but not Swing. The CDC is upgradable with different other configurations like a RMI configuration for RMI support.

4.2 Java Versions for PDA’s

For PDA’s, we find a lot of Java implementations on the market that vary a lot. Which support of Java can be found depends mostly on the choice of the operating system.

For WinCE/PocketPC, there exist only J2ME implementations that support the full CDC. The less powerful configuration CLDC is normally not available for PDA’s because of the high amount of memory and computational power.

Using Linux as the operating system for the PDA, there exist a lot of distributions for the iPaq like Familiar⁴ or mLinux⁵, where we found the full support of J2SE 1.3.1. This allows to easily port applications written for a PC to a PDA.

Another opportunity that exists at least for an iPaq is an operating system completely written in Java called SavaJE⁶. It supports its newest version the J2SE 1.4.1. Unfortunately, this operating system will be developed for the iPaq no longer but only for cell-phones.

4.3 Java on the Programmable Lego Brick (RCX)

For the RCX, there exists a Java implementation that is available as open source: Lejos⁷. Since it has nothing to do with J2ME, it does not support any standard but is a small and handy Java implementation where the virtual machine just needs 17kb. It is focused mainly on controlling the motors and sensors of the RCX with the well-known Java action-event model. Additionally, it supports threads which are very helpful doing robotics with Java, enabling the programmer to have several threads for controlling the movement of the robot and reacting to the environment.

5 The Framework for Integration of Smart Devices

This framework should take into account the heterogeneous field of smart devices by being on the one hand as flexible as possible and on the other hand supporting the user as much as possible.

Looking at Figure 1, which is a UML representation of the proposed framework, it is obvious that the framework follows a device centered approach. The class that all other classes are concerned to, direct or indirect, is the Device class. In the next section all used classes will be explained and their connections with each other will be made obvious.

⁴http://familiar.handhelds.org
⁵http://www.lisa.de/pages/de/mlinux/mlinux.html
⁶http://www.savaje.org
⁷http://lejos.sourceforge.net
5.1 The Device Class

The Device class as the most important one of the framework will be implemented as an abstract class since it doesn’t make any sense to instantiate it. It just encapsulates very general information about a device, like the control degree for the device and usecases the device should be used in. The control degree of a certain device could vary a lot depending on the device. The two major levels we want to tackle are programmable devices, those are devices we can control directly with self written programs like PDA’s or the Lego RCX brick. On the other hand there are devices we can only exchange data with like digital cameras, a c-Pen or even a mobile phone (at least most mobile phones used today are still not programmable). Therefore we have foreseen two classes that inherit from the Device class, ProgrammableDevice andDataExchangeDevice.

Furthermore the Device class gives a user the possibility to register himself as a listener for events that occur at the device. Those so called SmartDeviceEvents encapsulate different information about what happened at the device. For example each event got a certain name, a type and a result that the user can use to process the event.
5.2 The ProgrammableDevice Class

This class should model a device we are able to write programs for. This might for example be a handheld or the RCX brick from Lego. One important information for these kind of devices is whether we can only send programs to them or if it is possible to interact also with these programs what is important if we want to get data back from the device at runtime. Next to the methods needed to control the interaction mode we also need methods for controlling the program on the device, like a method for uploading a new program or running it.

5.3 TheDataExchangeDevice Class

Devices that are only capable of collecting data and providing this data afterwards should be modeled by this class. Here we have to divide these devices by the way they are providing the data after collecting it. For example would a digital camera typically be mountable in a usual filesystem and the memory of the camera could be handled as a normal drive. Other possibilities could be the data exchange over a database, a network connection or as a Java object. A method that this class has to provide is to access a handle to the data provided by the device. In the case of data exchange over a filesystem this handle could for example be the path the data is put in, or in the case of a database this handle will be the url of the database together with the account information for this database. To allow a wide variety of handles the method that gives success to it has Object as a return value, just to ensure that every Object could be returned.

5.4 Classes Needed by the Device Classes

The next sections will explain what kind of classes are needed by the Device classes.

5.4.1 The UseCase Class and the Action Interface

The UseCase class should model a usecase the special device is used for. Therefore it foresees actions that must be executed to fulfill this usecase. Every action will have to implement the Action interface that guarantees a method called execute which returns an Object that represents the outcome of the fulfilled action.

5.4.2 The Program and the SourceCode Class

For the ProgrammableDevice class it is necessary to encapsulate a program that should be deployed on the device. This is done by the use of the Program class. Within this class we might select three different type of programs: the program as a source code, a Java object, or an executable. As an additional information this class also encapsulates the information how to reach a compiler for the sourcecode. In a later version it might also have the possibility to directly compile the source code with the help of the compiler. It is possible to compile your favourite software or any application for the target(PDAs) on your PC. While this has been a difficult to accomplish task in the past, you can do it much easier today. You just need GNU/Linux and some of its nice little tools, most
importantly toolchain-source and dpkg-cross. There are some cross-compilers available for JAVA in the market too. One of them is FastJ, it is a Java language cross-compiler developed especially for embedded systems developers. But since it is still open what the return value of this kind of method would be we are also thinking about some factory class to solve this problem.

![Figure 2: An UML instance diagram for an example scenario](image.png)

Figure 2 shows an example scenario with a Lego mindstorms robot that communicates via infrared with an iPaq handheld. The iPaq handheld additionally receives images captured by a camera that is mounted on the robot.

## 6 Communication within the Framework

As already described in the section about the different Java versions for smart devices, the possible network communication possibilities may vary a lot. Sockets for example should be supported by every Java version except Lejos. But RMI will not be supported by every J2ME configuration. In the case of Lejos there exists a HttpProxy that directly ships with the package and allows to run servlets on the RCX.

Since developers using the framework presented here should not have to fiddle with the different network communication architectures our framework should handle at least Sockets, RMI and HTTP request transparently. Solving this problem we will implement a special communication layer that automatically notices what kind of communication is available for the device and that afterwards communicates in a transparent way with the framework. This could for example be done by implementing a special interface that has actually three different implementations, one for each communication architecture.

Another approach would be to write a proxy class that acts as a broker between the different protocols. A broker like this could easily be implemented using the MatchMaker\[4\] framework which is a communication framework for synchronizing Java applications. Going this way means also to create certain profiles for each device that for example indicates what kind of protocol the device supports. Those profiles could be compared to a description language for smart devices. One approach in this direction might be to use the Device
Description DTD from SyncML\textsuperscript{8}. Even if the authors consider this description not rich enough at this point in time. In any case will a description language like this today be in XML.

7 Outlook

Still an open and very important question is how to reach a smart device that is connected to a computer. Since the Java support for interfaces like a USB or the serial port is almost not existant this might lead to some problems towards auto recognition of connected devices. Therefore we will most likely also implement a factory class close to the SpaceAccess class of the JavaSpaces framework\textsuperscript{5}. The problem is very much equal to the one in JavaSpaces since there also exists no multicasting mechanism to ask for all reachable JavaSpaces.

Also the problem that different devices talk in different protocols over the same interface is not yet tackled. For example does a c-Pen need IRDA support for infrared communication while a Lego RCX brick has his own protocol for the infrared port.

Conversely the control of smart devices over HTTP should be handled transparently by the framework. One does not need to be always in front of the RCX. This could be achieved by the medium of designing some applets which communicate through a mediator called a proxy and in turn proxy will pass the messages to the RCX.

References


\textsuperscript{8}http://www.openmobilealliance.org/syncml/docs/syncml_devinf_v11_20020215.pdf