# Towards Real World Object Orientation

Paul Holleis, Albrecht Schmidt

Research Group Embedded Interaction
University of Munich, Germany
{paul, albrecht}@hcilab.org

Creating physical user interfaces is by no means an easy task. It involves efforts from many areas like HCI, design, engineering and application developers. However, writing programs that use physical devices is often hindered by the gap between them and the programming environment that developers are used to. We suggest to simplify programming and working with devices like small embedded systems, alarm clocks, mobile phones, and so on by enhancing the Eclipse development environment with plug-ins that encapsulate the integration of such devices in a familiar way. Real world objects can then be used in an intuitive way together with the power and possibility of standard programming languages. This shields developers from details and complexity as they do not have to learn new types of programming or syntax they are not used to.

## Introduction

Research in human-computer-interaction, design and other fields has come up with many devices that have communication capabilities with technologies like IrDA, Bluetooth, GPRS, UMTS, and others. Input and output devices are especially designed for a certain or general purpose. One of the main problems of integrating them in programs and systems is the difficulty of interfacing those to conventional programming languages that are used by application developers. Recently several projects emerged helping developers of ubiquitous applications in using embedded devices in their programs. Most of these are especially designed for rapid prototyping and to protect developers from core Java, C or Assembler programming.

An example for such a project is the Equip Toolkit [1] that allows combining graphical representations of devices and methods to form an executable data flow graph. This is very convenient for quick prototyping; however there must be a huge set of very general functions to be able to ensure at least minimal programming power. For more complex applications this soon gets complicated and tedious.

Another approach, for example taken by the creators of the PLUE programming language [4], is to enhance an existing programming language (JAVA) with special constructs. PLUE enables the quick insertion of event handling rules. However, it requires users to learn a special syntax to specify these rules.

Tim Kindberg et al. propose in [5] to associate “people, places, things” with a URL to create a connection to the physical world. They opt for using the web as middleware to connect the virtual and the physical. Besides a wide deployment and a

range of accepted standards, a main argument is the ease of creating intuitive, nice user interfaces easy to understand for the average user. We emphasise more the support of developers who do not need the generous possibilities of web browsers as user interfaces but want simple-to-use interfaces to access all possible aspects of physical objects.

Most of these approaches share the weakness that application developers wanting to go beyond mere prototyping have to learn entirely new constructs or ways of programming. We suggest completely incorporating external devices into an integrated development environment (IDE) that developers are already used to. All aspects and features of this IDE like modelling, code completion, debugging and so on can still be exploited and calls to external physical objects are treated as calls to any precompiled resource.

## Development Support

There are many generic IDEs available like MS Visual Studio or NetBeans. Even more can be found for specialized tasks and languages or proprietary compilers. We chose Eclipse [2] as development platform since it is widely used, open source and open for integrating plug-ins that can extend virtually every aspect of the platform.

As controllable devices, we chose TecO Particle Smart-Its [3] as a starting point because they are small, have a powerful microcontroller, communicate wirelessly and are easily extendable with all types of sensors and actuators. There exist interfaces to the communication protocol in many languages like C, C++, C# and Java.

We provide an Eclipse plug-in called TROOP (“Towards Real-world Object Oriented Programming”) that integrated itself as a view into the IDE. Figure 1 shows a screen dump of the Eclipse IDE. In the bottom right corner, the TROOP view is activated. The plug-in automatically searches for Particles. In the example, it finds one (with the ID *1.229.0.0.0.0.180.107*) and lists 5 methods exposed by this Particle. A class named *Particle_1_229_0_0_0_0_180_107* is automatically generated by the plug-in in the background and appears in the project explorer. In the code window it can be seen that the only step needed to be able to call one of these methods is to get an instance of this class (*particle*). This object can then be used without the developer needing to know any implementation details. IDE features like code completion (e.g. method names and parameters, exception handling) and debugging are enabled as if it were a native method invocation. In Figure 2, a second Particle equipped with a display has been switched on. Methods to access this display are immediately available in the Eclipse environment for simple use.

### Implementation Details

The moment the Eclipse plug-in is activated, it broadcasts an interface description request. All Particles in RF range that have been configured to listen to such requests will immediately answer by sending an interface description of all methods that they provide to externals callers. The same happens when a new Particle is switched on or comes into reach. This particle then receives a specifically addressed request.

To minimize data transfer, we do not use a sophisticated standard IDL (interface description language). We have implemented a very concise protocol currently supporting parameters of basic types only. For each method, the following data is sent (the numbers in parentheses indicate the number of bytes associated to each field):
*<methodId (1)> <returnType (1)> <methodName (≤20)> <parameterType (1)>**

This imposes some limits on the number of available methods, the length of their names, etc. but keeps the average packet size down to approximately 14 bytes per method. When the Eclipse plug-in receives such interface data from a Particle, it automatically generates (or updates) a Java implementation specific to that Particle. According to the proxy design pattern, this class delegates method calls from the main programming task to the Particle and returns its answers. The proxy interprets method parameters according to the specified types and sends an addressed remote procedure call to the Particle associated with the current class. Those calls are sent in a similar data format as the interface description using acknowledged traffic. Thus, if a message is not received or a method with a non-void return type does not return a value after some timeout, an exception mechanism can take appropriate actions.

## Some Open Questions and Future Work

We have described how to seamlessly incorporate access to physical objects into a well known IDE. A main argument in favour of this technique is that programming is in no way altered or different from the usual programming experience. There is also no restriction to mere prototyping. In addition we argue that, other than end-users, developers do not need fancy graphical user interfaces (although the plug-in can be extended in that way) but want to see clean interfaces. Because of the physicality of the objects under control, most methods are self-describing (like for example "*switchRedLedOn()*"). An important difference to most distributed systems is that devices can enter and disappear or be switched on and off dynamically. However, the setting does very probably not need highly sophisticated methods used with extremely dynamic networks. Additionally, physical interaction with the objects at hands is possible and must be taken into account.

Currently we are working on several important additions: Events must be added to the set of methods a device can expose. Applications can then register methods to be called whenever such an event is raised. It should be possible to add basic documentation to methods and we are looking into how to connect to different types of objects and are currently testing Bluetooth together with mobile phones and PDAs.

Some of the interesting questions that arise are: What exactly happens if an object is no longer there or active? Can one object take over the role of another when it has (at least) the same capabilities? Can this be modelled by adding inheritance to the system? And how far does pushing the OOP paradigm further in this direction make sense (abstraction, polymorphism, inheritance)? Is the ability to refit (to program) objects with new or additional capabilities important and possible in that sense? Is it sensible to deploy interfaces on a central server or (in line with the "web presence" mentioned in [5]) associate a web page with each particle (tagged with a version) to enable reusing interfaces and using unseen objects?

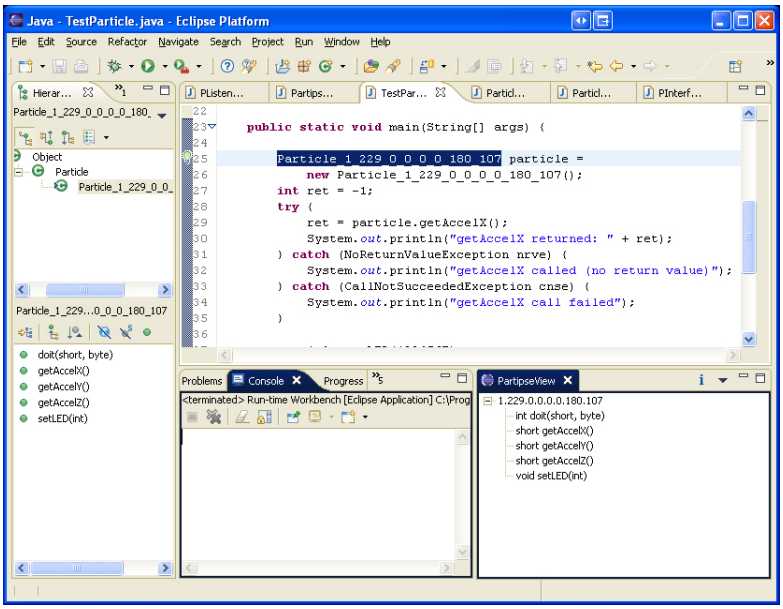


**Figure 1**: The Eclipse development environment. The view of the plug-in is displayed in the right bottom corner. It currently shows the interface of one particle. The code window shows how to call a method of this particle (including exception handling).

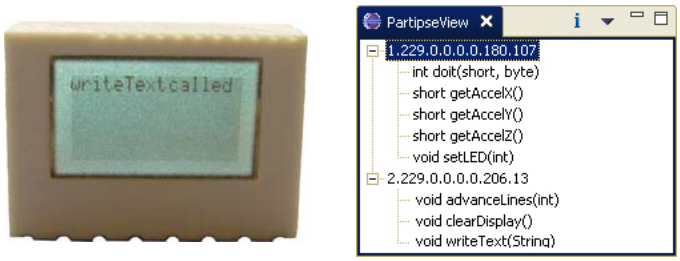


**Figure 2**: Another particle with a display has been activated. Its interface is added.

## Literature

[1] Greenhalgh, C., Izadi, S., Mathrick, J., Humble, J., Taylor, I. ECT: A Toolkit to Support Rapid Construction of Ubicomp Environments. In Adjunct Proceedings of The 6th International Conference on Ubiquitous Computing (UbiComp), 2004

[2] Eclipse Project Page, http://www.eclipse.org/

[3] Decker, C., Krohn, A., Beigl, M., Zimmer, T. The Particle Computer System. In Proceedings Proc. of the ACM/IEEE Fourth 4th International Int. Conference Conf. on Information Processing in Sensor Networks 2005, Los Angeles, USA.

[4] Cho, E-S., Lee, K-W. Security Checks in Programming Languages for Ubiquitous Environments, First Workshop on PSPT at MobiQuitous 2004

[5] Kindberg, T., Barton, J., Morgan, J., et al. People, Places, Things: Web Presence for the Real World. In Journal of Mobile Networks and Applications, Vol. 7, Nr. 5, p. 365-376, Kluwer Academic Publishers, Hingham, MA, USA, 2002