P-NET Management with Java based Components

Martin Wollschlaeger

Abstract

The introduction of Java based software components is a challenge for developers and users of fieldbus products. The paper shows concepts, prerequisites and implementation aspects of Java based components providing management tools for P-NET. The mapping of management related functionality of a P-NET system to Java components is described. The influence of current specifications of the Java Application Programming Interfaces is discussed. Special attention is paid to the interaction between OLE-based software and Java components. Exemplary implementations for P-NET using VIGO are shown. Based on the implementations, the paper points out the integrative effects of the solution. The users' and vendors' benefits and problems of Java based software concepts in the area of fieldbusses are discussed. Future trends are shown, leading to an integration of different fieldbusses into a network centric process information system.

1. Introduction

There is no doubt, that Java is one of the most interesting trends in programming. The primary goal of Java is characterised by the slogan "write once, run anywhere" and means platform independence. This can be achieved by introducing Java virtual machines (VMs). Java coded programs (byte code) can be transparently used on any virtual machine(**Fig. 1**). This is, of course, a big step forward on the way towards single, uniform programming environments and solutions. On the other hand, the user has to pay for that goal - the Java byte code has to be interpreted by the virtual machines, which makes it slower than code running directly on the target machine.

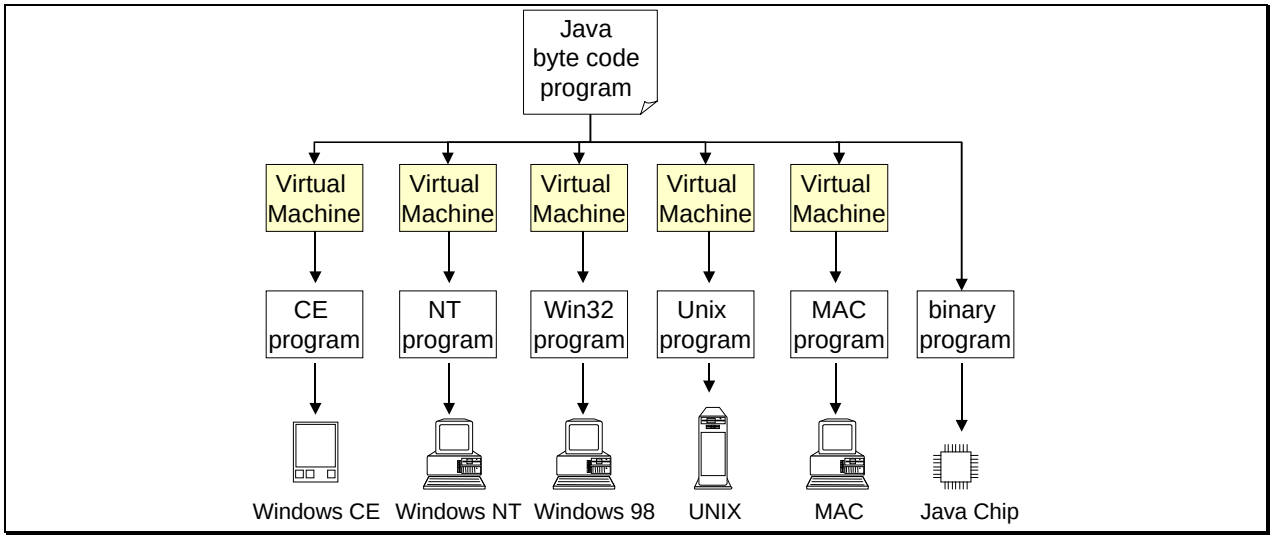

Fig. 1: Platform independence achieved by Java virtual machines

Dr.-Ing. Martin Wollschlaeger
Institut für Prozeßmeßtechnik und Elektronik (IPE)
Otto-von-Guericke-University Magdeburg
P.O. Box 4120
39016 Magdeburg
GERMANY

Tel.: +49 (391) 67-1 46 53
Fax: +49 (391) 5 61 63 58
e-mail: mw@ipe.et.uni-magdeburg.de

The platform independence is requested mainly by those users, who develop Internet-ready solutions. This explains the symbiosis of Internet and Java. However, as the Internet and its associated technologies become more and more into the focus of other application areas, including automation and control, the number requests for Java based solutions are steadily growing. As a result of this evolution, a growing number of application programming interfaces (APIs) for different purposes is defined /1/. They cover topics from database access, security and graphic representation, up to LAN-like directory and naming services, remote procedure calls and embedded Java. The Java enterprise API, consisting of naming services (JNDI) /2/, remote method invocation (RMI) /3/ and database access (JDBC) /4/ is very interesting for application in industrial communication systems. The existence of such an API shows the importance of Java and Internet technology for process control solutions.

Other key features of Java solutions are enhanced security based on the sandbox model, and the method of software distribution via downloads over a network. Both features are important for the use in automation and control systems, as well. Especially methods for distribution of software are considered absolutely necessary in distributed environments, like industrial communication systems (**Fig. 2**).

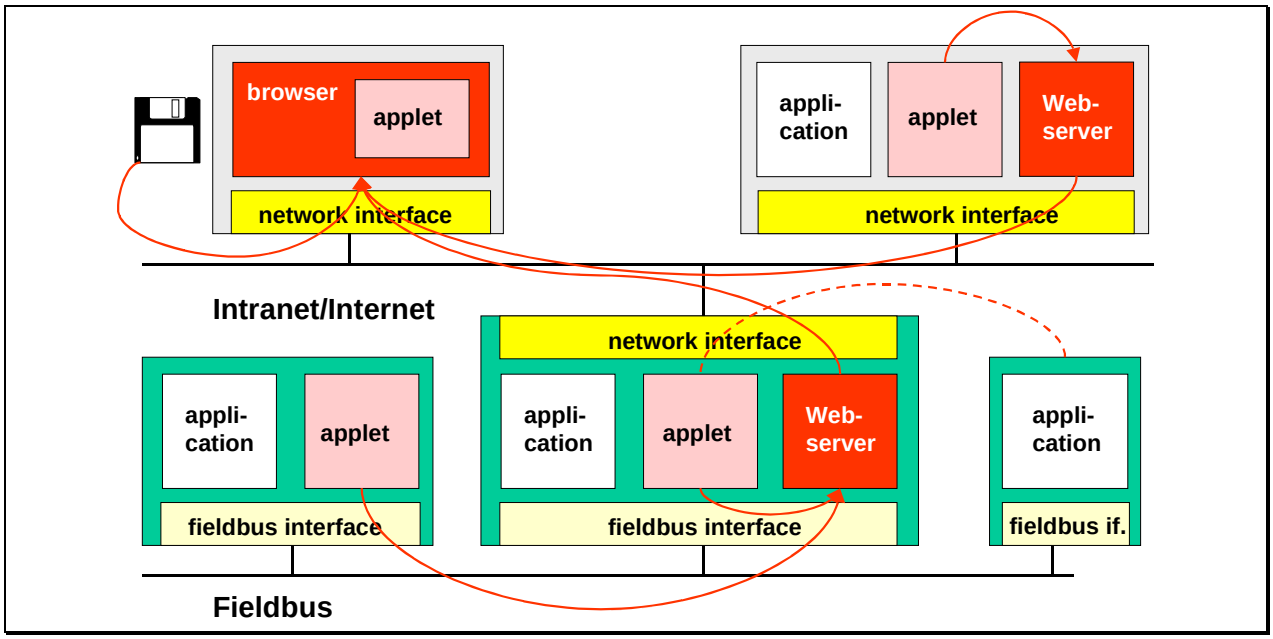

Fig. 2: Methods of software distribution using applets

2. Scenarios for the implementation of Java based components

Corresponding to the software distribution methods shown in **Fig. 2**, different scenarios for the implementation of Java applets and applications are possible (**Fig. 3**).

Java-Applets

Java applets (a) are the most common way of Java components' implementations. They can be used in combination with suitable container applications, e.g. visualisation packages. Relying on Java's features for graphical and textual data presentation, applets provide a user interface. The interaction with the user is handled by events. Java applets are an interesting alternative for developer of HMI software and SCADA packages to prevent re-development of software. The programming cycle is simplified, object-oriented software components can be designed. As a side effect, the interfaces appear to be uniformed, even for different environments and application platforms. This technology supports the idea of providing small components fitting into a framework. In addition, using standard APIs, existing solutions can be reused.

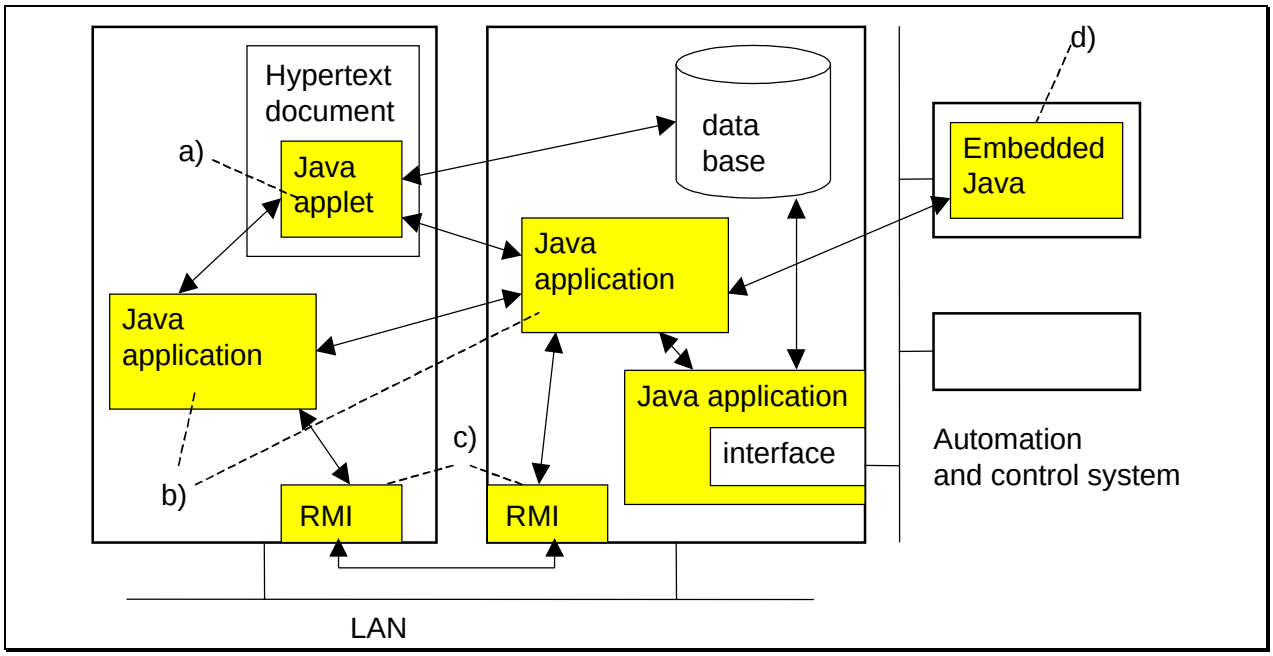

Fig. 3: Different possibilities for implementation of Java components

A very special attention has to be paid to applets because of the increasing use of Internet browsers like Netscape Navigator or Microsoft Internet Explorer as containers for HMI applications. A large number of companies is providing browser based HMI solutions, that implement applets for visualisation and manipulation of process variables. The trends towards component software, Intranets and network-centric views is going to increase the meaning of applets in future. The Java beans /5/, special applets, are the opposite of ActiveX controls.

Java-applications

Using the development tools for Java, stand-alone applications (b) can be developed. These applications typically implement tasks like data analysis, recipe management and database connectivity, all tasks that are independent of the underlying operating system. In addition, Java applications can act as containers for applets, as well.

Java and distributed systems

The use of networked resources in Java is mostly implicit, without integrating the user into the management of the network and its access methods. However, as a network-centric system's approach, Java provides APIs for an explicit network use (c). The Java Remote Method Invocation API (RMI) defines the communication methods between networked components. The Java Naming and Directory Interface (JNDI) provides LAN-like management functions based on directory services. Especially interesting for heterogeneous networked systems is Java IDL /6/, that guarantees interoperability of those resources using CORBA (Common Object Request Broker Architecture) /7/. A complete ORB (Object Request Broker) written in Java is included in this package.

Embedded Java

Last but not least, embedded Java (d) /8/ could be a trend for the future of automation and control components. Based on a Java chip and an appropriate operating system, thin client architectures provide the basic part of a decentralised, networked component performing a special task like sensors or actuators. The programming of the component is done directly in Java, the programs are distributed across the network. An actual development pointing into that direction is the definition of JINI.

3. Java based components for fieldbus management

While the scenarios using Java applications, RMI and Embedded Java require a virtual machine running on the networked component, the applet scenario is different. Of course, an applet needs a VM as well, but the idea behind that scenario distinguishes from the others. The applet is not designed to interact directly with the fieldbus data, like an embedded solution. It is, at least in most cases, designed to map the management functionality to a user interface. This user interface runs within a container application, for example a web browser. The web browser includes the virtual machine and does not necessarily run on a device physically connected to the fieldbus.

Using the features of Java, an applet should map a more generalised, functional oriented management task to a user interface component. Typically, function blocks provide exactly that generalised view. This makes the applet's functionality independent from the fieldbus used as a transport layer. As example, an applet could be implemented for manipulating the values of a PID controller. The web page hosting the applet on the client then submits to or gets the values from the applet. A script running on the client with the browser or on the server is required to interact with the fieldbus interface card or with the fieldbus' database. As explained above, no virtual machine is necessary on the server. The interactions can be performed via traditionally interfaces like COM or OLE Automation. **Fig. 4** shows the principle structure of such an implementation.

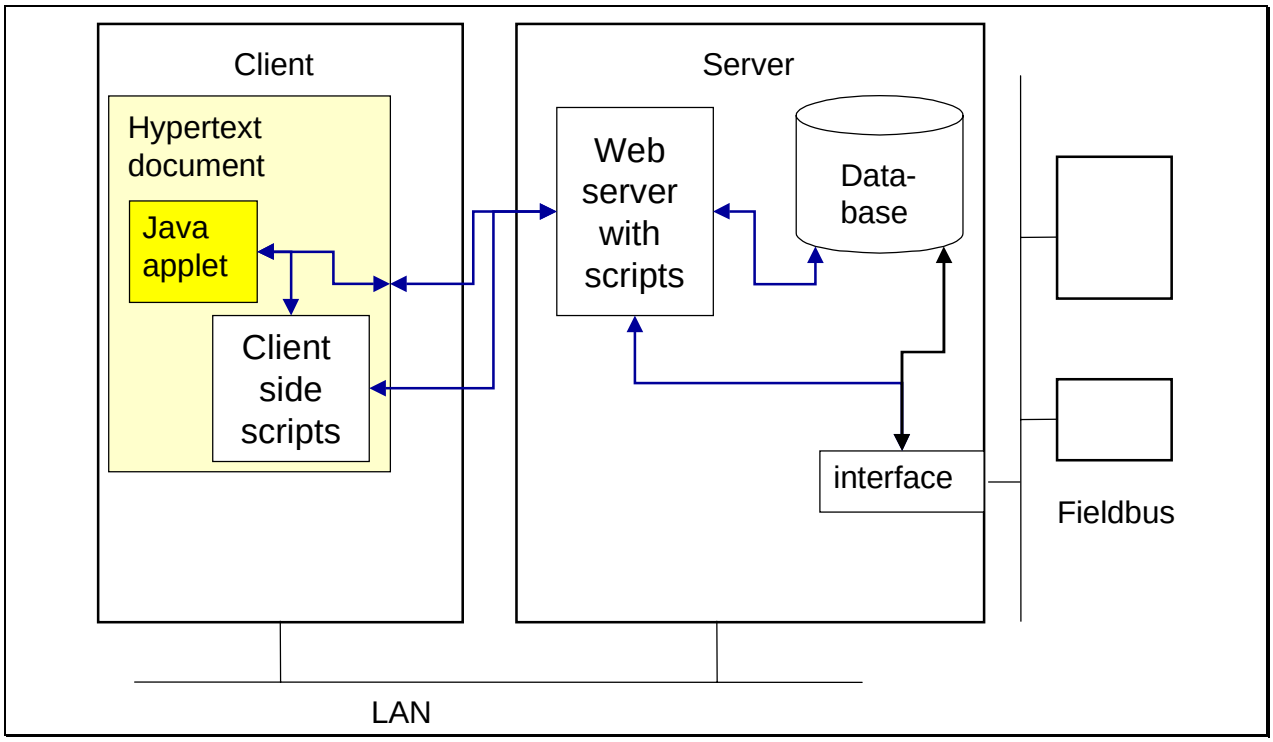

Fig. 4: Principle structure of fieldbus management using applets

The approach using RMI and Java applications requires a virtual machine on the server (**Fig. 5**). The server has to provide an interface between Java methods accessible through RMI and the fieldbus interface. Different solutions for such an interface are suitable. The simplest one is to use the fieldbus database. This method can be ineffective, because of two different accesses that have to be synchronised. Furthermore, it requires the existence of a Java interface to the database. However, this should not be a problem, if a standard database system is used. Another method is implementing the interface in a script, that translates the Java request to a dedicated function in the fieldbus interface. This requires a special scripting system to be installed. The most effective way is a direct coupling of the Java application with the interface, with other words a Java capable interface driver. However, platform independence of Java is concerned

with some problems when hardware, such as interface cards for fieldbusses, need to be accessed. Special drivers are necessary, implementing a data exchange method, that Java can handle. A solution for this is to split up into the interface into an implementation independent interface part, and a specific driver part (**Fig. 6**). While the interface part is pure Java, the driver part is implementation dependent and can use "native" operating system techniques like COM/OLE /9/, OPC /10/ or any other specific drivers for data access. So the different software strategies are supporting each other.

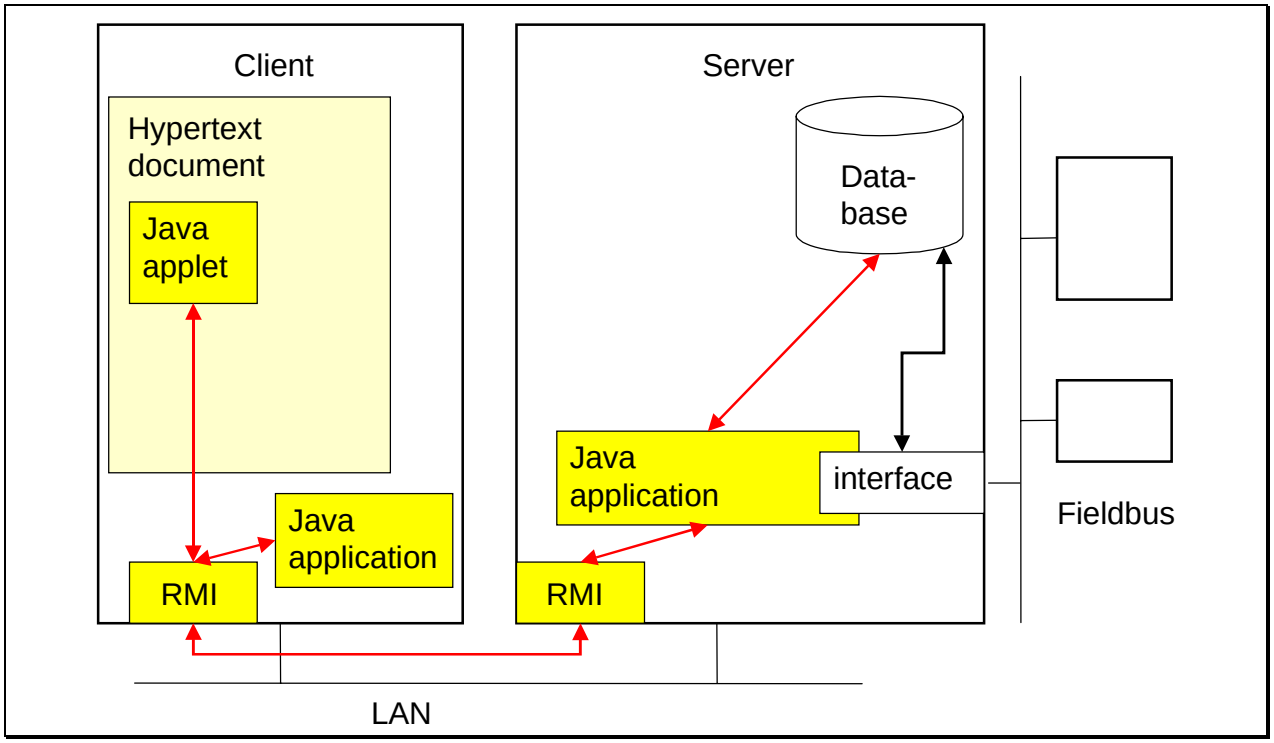

Fig. 5: Principle structure of fieldbus management using applications and RMI

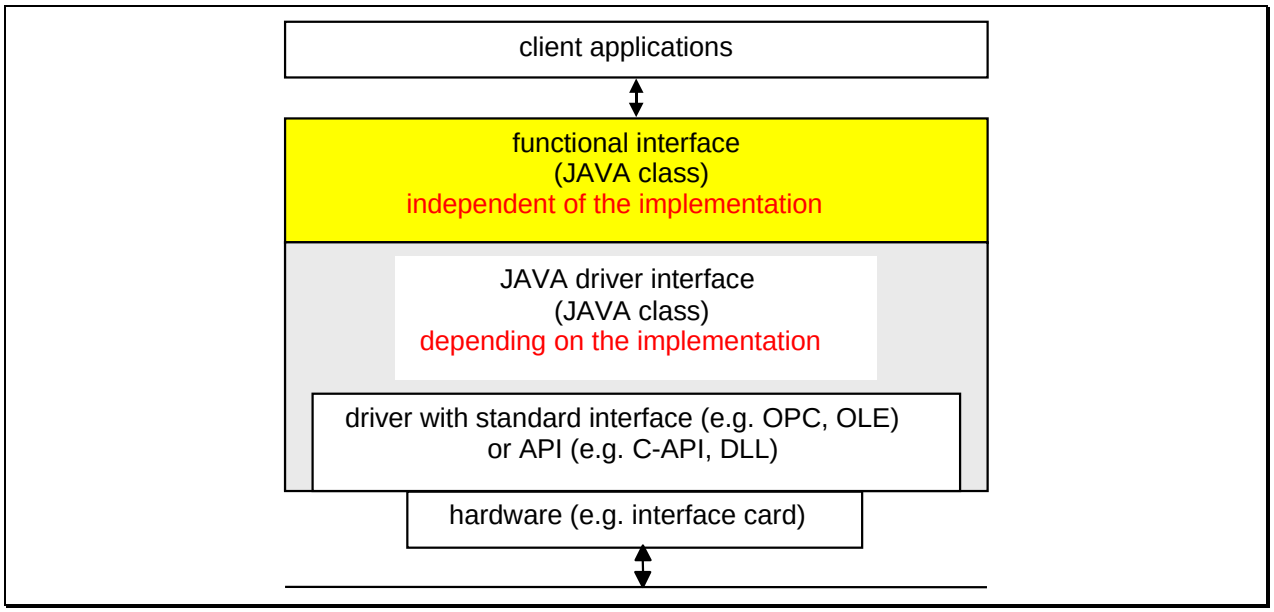

Fig. 6: Java based data access

4. Mapping of P-NET management functionality to Java components

The management functions found in P-NET systems can be distinguished into two major groups. The first one is the management of the topological system, the second one is the functional management. The topological system's management includes functions like adding, editing and removing of projects, nets, and modules. There are a lot of existing tools for that. These tools are part of VIGO /11/. A mapping of these tools' functions to Java based components is possible. However, it has to be considered, if such a mapping makes a sense. It is useful in applications, where platform independent management is required. The mapping itself can be implemented by accessing the COM interfaces provided by VIGO. This requires an installed VIGO on the machine, where the mapping is performed. So typically the server is used for that. A Java application or applet can access COM objects and can pass information to these interfaces or can invoke methods. The interaction of Java and COM can be achieved by generating COM wrapper classes, that manage the data transfer. Microsoft's Visual J++ supports this feature since version 1.1, a tool named JAVATLB /12/ generates Java class files from a given type library (**Fig. 7**).

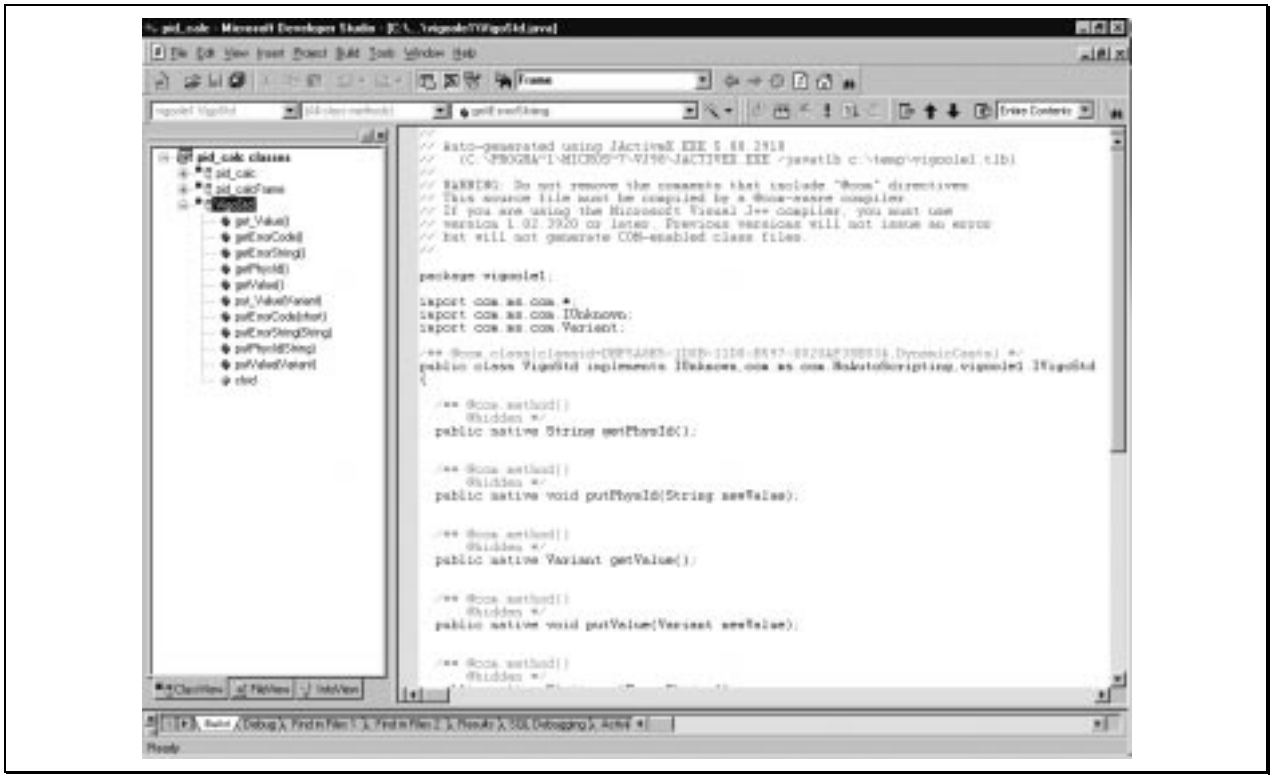

Fig. 7: Screenshot of the auto-generated Java class for VigoStd

The generated Java class provides the functions and data types, that can be used in a Java program. A generalised management tool can pass its calculations to the generated Java class in order to put this information into the VIGO database or directly into the modules. Thus, an application scenario according to **Fig. 5** has to be used, where the Java application on the server gets inputs directly from its user interface, via the RMI, or from a server-side script.

The functional oriented management tasks are usually common to different fieldbus systems. Based on function blocks, an applet provides a user interface for management tasks. The inputs and outputs of that user interface are linked to a fieldbus depending interface layer, that maps the applet's parameters to the data structures of the underlying fieldbus system. An appropriate place for implementing a script would be the web page containing the applet. So re-using of an applet is possible, while the fieldbus-specific mapping has to be done by generating a script for every supported fieldbus. However, using an applet as a user interface guarantees a consistent, functionally independent interface between the applet and the script. In P-NET systems, the script

could contain code, that uses the COM interfaces of VIGO (**Fig. 8**). The generalised structure of such a solution is shown in **Fig. 4**. As a conclusion, an applet based solution is possible, and is an interesting approach for platform independent frameworks. Especially in combination with web pages containing scripts this scenario is applicable.

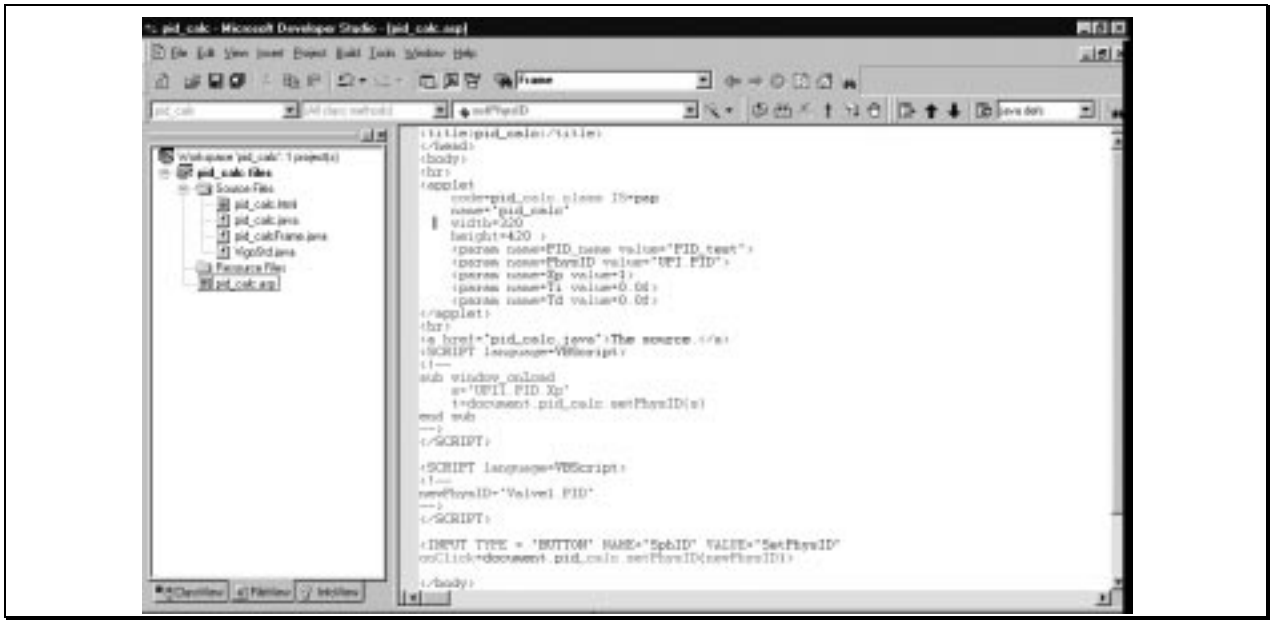

Fig. 8: Screenshot of a script managing an applet

5. Exemplary solution

An exemplary solution has been developed focusing on the functional oriented mapping of management tasks. **Fig. 9** shows, how easy an applet can be developed.

Fig. 9: Screenshot of source code and applet

This applet builds up a simple user interface for a PID controller. It reads the values of the PID algorithm from a selected channel and displays them. The interaction with the P-NET is performed using scripts. **Fig. 10** shows the relations between the components. The applet provides functions for setting and retrieving the values of the internal parameters. These functions can be called by a script. The internal algorithms of the applet handle the values and check them for

errors like parameters out of range and so on. Based on the theory of control, the applet could do much more. There could be tuning algorithms, graphical displays an interaction and so on. This would lead to a generic applet for tuning the PID parameters.

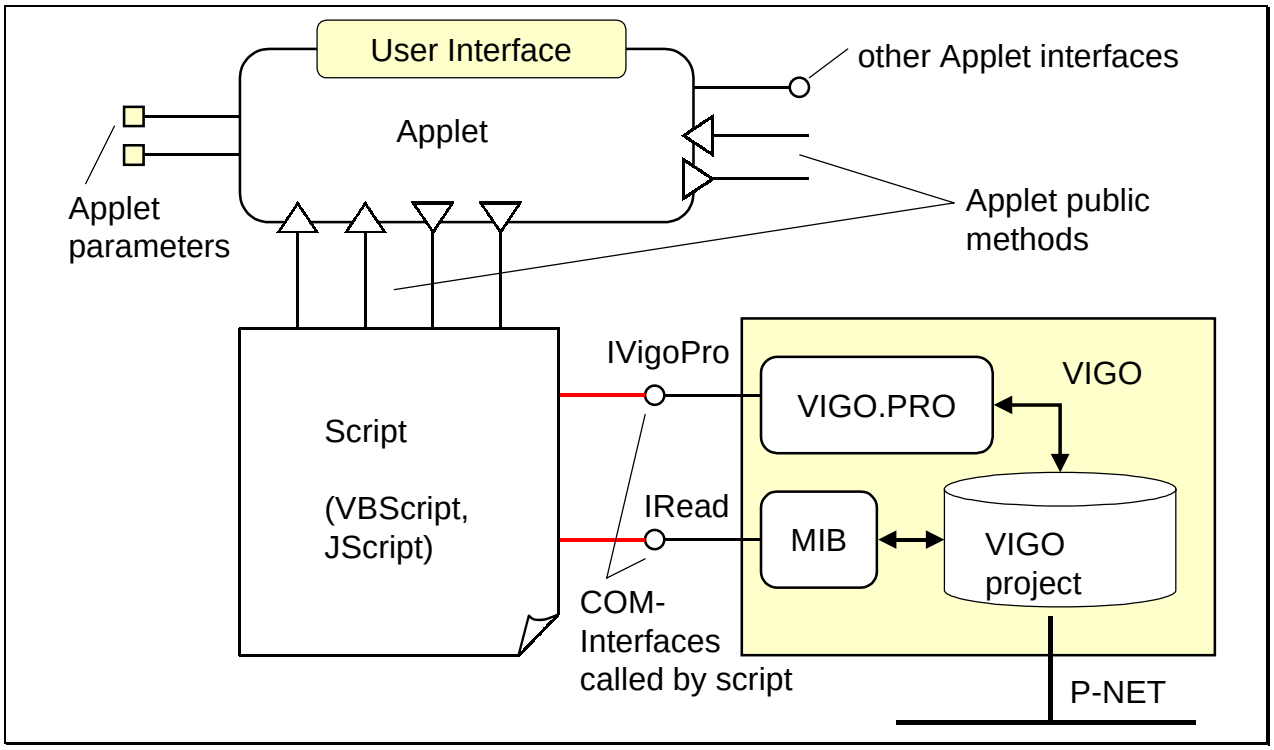

Fig. 10: Relations between an applet, a script and VIGO in the P-NET example

In order to build up the interaction with P-NET, some special functions are added to that applet. This has been done by inheritance of the generic applet. When the applet shown above is initialised, it reads the parameter "PhysID" as the reference to the underlying P-NET system. This parameter should refer to a PID channel. The applet checks the channel type for verification. The used PhysID is displayed for information, then the script functions are invoked to read the values of the channel. The applet provides the correct PhysIDs generated from the channels PhysID and passes them to the script. The script uses the COM interface of VIGO and gets the values. Via properties (public methods) of the applet these results are transferred to the applets algorithm.

Using the channel concept in P-NET, the applet is able to handle all the PID channels in a given project. A list box containing all PhysIDs of PID channels in the project can be built up during initialisation of the applet, or by clicking the appropriate button. To generate the list's contents, applet and script relate on each other as described above.

6. Summary and future trends

The example shows the feasibility of Java applets for management tasks of P-NET installations. However, it has to be considered, whether an applet is necessary or an ActiveX-control can be used. Because of VIGO as a COM based system, it might be easier to generate and use ActiveX-controls. The benefits of using Java solutions lies in the platform independence and in enhanced security, supported by specific APIs /13/. If there is an existing framework in Java, the described solutions allow re-using previously developed software. In combination with scripting powerful solutions can be built.

Another interesting trend is the further development of the Java chip. Devices based on this chip are expected to be available within a short period of time. This would give Java solutions a big

support. Finally, the existence of Java virtual machines on programmable networked devices, whether using Java chips or traditionally controllers, would allow pure Java solutions and devices directly programmable in Java. This is a big step forward to re-usable software and enhanced security, but, on the other hand, those solutions would require more computer power to reach the calculation speed controllers have now.

References

/1/ http://java.sun.com/products/api-overview/index.html

/2/ http://java.sun.com/products/jndi/index.html

/3/ http://java.sun.com/products/jdk/1.1/docs/guide/rmi/index.html

/4/ http://java.sun.com/products/jdk/1.1/docs/guide/jdbc/index.html

/5/ http://java.sun.com/products/jdk/1.1/docs/guide/beans/index.html

/6/ http://java.sun.com/products/jdk/idl/index.html

/7/ http://www.omg.org/corba/

/8/ http://java.sun.com/products/embeddedjava/

/9/ Brockschmidt, K.:
Inside OLE. Second Edition.
Microsoft Press, Redmond, 1997.

/10/ n.n.:
OPC Data Access Automation Specification, Version 2.0
OPC Foundation, October 14th 1998.

/11/ Cramer, O.:
VIGO TOOLS.
4th International conference on the P-NET Fieldbus system.
Oporto, May 2nd -3rd 1996, proceedings.

/12/ Ladd, S.:
Active Visual J++.
Microsoft Press, Redmond, 1997.

/13/ http://java.sun.com/products/jdk/1.1/docs/guide/security/index.html