AC 2012-3033: APPLICATION OF JAVA TECHNOLOGY IN INDUSTRIAL REAL-TIME SYSTEMS

Dr. Javad Shakib, DeVry University, Pomona

Application of Java Technology in Industrial Real-Time Systems

Industrial automation is currently characterized by a number of trends induced by the current market situation. The main trends are the pursuit of high flexibility, good scalability, high robustness of automation systems, and the integration of new technologies in all fields and levels of automation. Of special interest is the integration of technologies into the control area.

In this context, object-oriented languages have gained importance. *In particular,* there is an increasingly growing interest in Real-Time Java because of its challenges and its potential impact on the development of embedded and real-time applications.

Java applications are executable on every platform that can run a Java Virtual machine, JVM. Together with typical concepts of object-orientation, Java opens many possibilities for a reusability of the code. Java also provides high stability of applications, which makes it an interesting programming language for control engineering.

Although the creation of a uniform specification for real-time behavior in Java is vitally important right against the background of platform independence, it was not possible to find a common approach of all parties thereto. Currently, there are suggestions for the enhancement/modification of the standard-Java-specification by two consortia with the goal of making Java applicable to real-time systems.

The suggested real-time extensions for the Java language are: Real-Time Specification for Java (RTSJ), developed by the Real-Time for Java Expert Group under the auspices of Sun Microsystems; Real-Time Core Extensions (RTCE), developed by the Real-Time Java Working Group operating within J-Consortium; and finally the Real-Time Data Access (RTDA) specification, developed by the Real-Time Data Access Working Group (RTAWG), and operating within J-Consortium.

These specifications are compared together. It can be stated that all of them follow different approaches but also focus on different application areas. The RTSJ and RTCE have the goal to provide a more general way of enabling Java to become usable in the real-time domain, while the RTDA focuses on the special application field of industrial automation and understands real-time capabilities only as one of several requirements, which are typical for this area.

The use of Java for control programming at the field level shows advantages, especially in systems where Java is also used for non-real-time applications. For implementing such systems, special attention has to be paid to restrictions regarding the mutual interactions of the control application running under real-time conditions and the non-real-time application.

If the real-time Java platform provides support in this respect, this can be realized very easily. However, most real-time Java products and specifications do not follow the requirements of industrial control applications; hence, special attention that has to be paid to these aspects is discussed.

This paper is to present a practical teaching module that introduces and exposes Java programming techniques to electronics engineering technology in a junior-level course. It also attempts to serve as an innovative way to expose technology students to this difficult topic and give them the fresh taste of Java programming while having fun learning the Industrial Applications.

1. Introduction: New Programming Paradigms in Industrial Automation

Industrial automation is currently characterized by a number of trends driven by the current market situation. The main trends are the pursuit of high flexibility, good scalability, and high robustness of automation systems; the integration of new technologies; and the harmonization of used technologies in all fields and levels of automation. Of special interest is the integration of technologies that were originally developed for the office world into the control area. This trend is characterized by the emergence of industrial PCs, operating systems for embedded devices like Windows CE, embedded Linux and RT Linux, data presentation technologies like XML, and communication technologies such as Ethernet and Internet technologies[1, 7].

In this context, object-oriented languages such as Java have gained importance. Java, which has evolved with the Internet and related technologies, meshes well with different areas of industrial automation as well as enterprises [1].

This paper presents the development and application of a practical teaching module created at DeVry University that introduces and exposes JAVA programming techniques to electronics and computer engineering students well before they learn any of its applications in a junior-level course.

Before our course, students have studied only Microsoft Visual Studio-based C++ programming with basics of Algorithm design and basics of computer architecture. First of all, we teach them Java with command line interface at the beginning and then GUI. The first part is not attractive for students as they are not able to repeat compile command or edit source code without an IDE. The students seem to be afraid of command line (shell) and Linux as well. The second part is much more popular. Because of the need for Web API of controlled processed in the industry, they are exposed to real-time systems and they also rewrite Java application to Java applet.

Mechatronics, Control Systems, and Industrial Automation are essential senior-level courses that expose students to a wide variety of fields of OOP applications.

This paper attempts to explain how this learning and teaching module is instrumental in progressive learning for students by doing these computations by leveraging the power of Java. It will serve as an innovative way to expose technology students to this difficult topic and give them the fresh taste of Java programming while having fun learning the Industrial Applications.

In the following sections, the boundary conditions, advantages, and problems of using Java in the area of industrial automation will be described in more detail. A special focus will be on the lower levels of the automation pyramid, where real-time requirements are of significance.

2. Requirements in Automation and Typical Application Areas of Java

Java can be characterized as a high-level, consequently object-oriented programming language with a strong type-system. One important feature is the idea of a Virtual Machine abstracting the concrete underlying hardware by translating Java programs to an intermediate language called Byte-code, which is executed on the Java Virtual Machine (JVM). Thereby, the concept "Write Once, Run Anywhere" (WORA) is realized, enabling a platform-independent application design. This means that Java applications are (of course, under consideration of different Java versions and editions) executable on every platform that can run a JVM. Together with typical concepts of object-orientation, Java opens many possibilities for a reusability of the code. Java also provides high stability of applications. This is realized by extensive checks (e.g., regarding type or array boundaries) during compile, load, and runtime. Since error prone concepts, like direct pointer manipulations are beyond the scope of the language and memory allocation (and de-allocation) is realized by an automatic memory management, the so-called Garbage Collection (GC), the efficiency of software development with Java, is very high. Although it is not easy to specify this in figures, some sources state at least 20% improvement compared to C/C++ [9]. In contrast to the most used PLC (Programmable Logic Controller) programming languages specified in the IEC 61131, the efficiency increases by more than 50%. The extensive, easy-to-use networking abilities of Java can help to reduce the difficulty of programming distributed systems.

Besides general advantages, the typical potential application areas for Java in industrial automation and their specific requirements have to be considered. On the upper levels of the automation pyramid (ERP and SCADA/MES-Systems), Java is already used as one alternative. The requirements in this area are very similar to typical IT-applications and characterized by powerful hardware. Here, Java2SE (Standard Edition) and Java2EE (Enterprise Edition) with a wide variety of APIs supporting different technologies for communication, visualization, and database access are the proper Java platforms. Examples for Java applications can mainly be found for different interface realizations of ERPs and the implementation of advanced technologies on the MES level [11].

At the field-device level, Java is still not a very common language as the requirements here are a problematic issue for standard Java versions. Many features of Java, normally responsible for typical advantages, cause problems at the field-device level. The hardware on this level is very heterogeneous and most devices have only limited resources with respect to memory and computing power. For communication purposes, several different field bus protocols are used, although the increasing relevance of Ethernet-based protocols shows possibilities for a common communication medium [15].

For indicator and control elements, Java provides powerful APIs for user-interface programming (ATW and Swing) and there are no special requirements regarding computing power. In contrast to this, for control devices such as PLC, Soft PLC, or IPC, there are constraints that do not fit in every case to the features of standard Java. These devices are characterized by:

- real-time requirements for application parts that realize control functionalities (cyclic control program execution with a defined cycle time)
- direct hardware access to the I/O level
- today's usage of PLC typical programming concepts (e.g., IEC 61131)

A special case are smart I/O-devices, where the computing resources are even more limited than in normal PLCs. Usually the PLC hardware is more attractive from an I/O and form factor perspective, but they hardly can mix and match traditional languages such as relay ladder logic with modern high level languages such as Java.

3. Problems of Using Java at the Field Level under Real-Time Conditions

It can be stated that for the use of standard Java in non-real-time conditions with limited resources, several problems have to be solved, even if the advantages mentioned above make Java an interesting programming language for control engineering.

Before describing the existing problems, the usual necessities for the design of applications in the field-control area will be given. A (distributed) control application requires, of course, real-time behavior as well as the realization of communication with other applications, field I/O, and/or remote I/O. Normally, hardware access regarding memory allocation, access to local I/O, and similar things are also necessary. Figure 1 gives an overview of the strengths and problems of Java against the background of these requirements. In general, Java provides advantages for tasks such as user interaction over an HMI or communication with other applications/remote I/Os over different protocols. The close relation of Java to the Internet world makes it easy to support communication via protocols such as http, ftp, or SMTP.

Unfortunately, other features of Java make it difficult to use it in control engineering without modifications or enhancements. This concerns the resource consumption of Java as well as the real-time capabilities and direct access to the hardware [6]. The main features of Java that make it difficult to use it in control engineering are Resource Consumption, Execution Speed and Predictability, Garbage Collection, Synchronization/Priorities and Hardware Access.

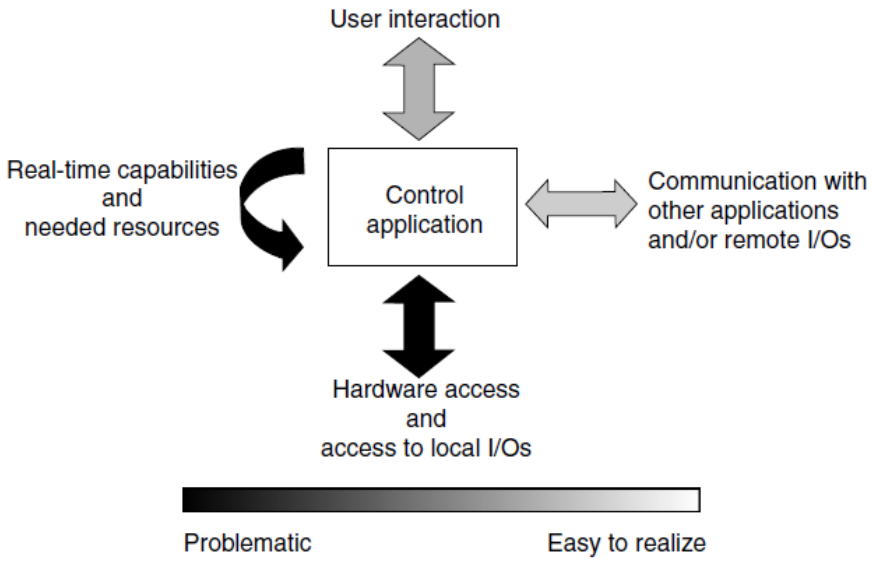

FIGURE 1. Requirements around a control application.

4. Specifications for Real-Time Java

Although the creation of a uniform specification for real-time behavior in Java is vitally important right against the background of platform independence, it was not possible to find a common approach. At the moment, there are suggestions for the enhancement/modification of the standard- Java-specification by two consortia. Both of them have a goal to solve the problems stated above and make Java applicable to real-time systems.

4.1 Real-Time Specification for Java

The first consortium is the "Real-Time for Java Expert Group," under the leadership of Sun Microsystems, which has developed the "Real-Time Specification for Java" (RTSJ) [2]. The formal scope for the development of this specification is the Java Community Process (JCP), where the RTSJ runs as Java Specification Request (JSR)-000001. The JCP defines a procedure that requires an internal and public review of the draft specifications as well as a reference implementation and a test suite called "Technology Compatibility Kit" (TCK). The JSR-000001 has reached the state of Final Release on 07 January, 2002, and a reference implementation has been developed by TimeSys Corporation [13].

General principles for the development of the RTSJ were to guarantee the temporally predictable execution of Java programs and to support the current real-time application development practice. For the realization of real-time features, the boundary conditions were set so that no syntactic extension of the Language needed to be introduced, and the backward compatibility was to be maintained by mapping Java Language Semantics (JLS) to appropriate entities providing the required behavior under real-time conditions. Furthermore, the RTSJ should be appropriate for any Java platform.

The more general WORA concept of standard Java is replaced by the so-called Write Once Carefully Run Anywhere Conditionally (WOCRAC) principle. This is necessary because the influence of the platform has to be taken into account. For instance, if the performance of the platform is not fast enough to meet some deadlines (this should normally not be the case, but may be, if the original platform was much faster), this is a serious problem that cannot be ignored.

In this context, the goal of the RTSJ is not to optimize the general performance of a JVM, but to improve the features causing problems for real-time systems like the GC, synchronization mechanisms, and the handling of asynchronous events.

To achieve this, the RTSJ enhances the Java specification in the following seven areas:

- Thread Scheduling and Dispatching
- Memory Management
- Synchronization and Resource Sharing
- Asynchronous Event Handling
- Asynchronous Transfer of Control
- Asynchronous Thread Termination
- Physical Memory Access

All these additions improve critical aspects in the behavior of Java or add typical programming features for real-time system development [10].

The RTSJ defines two low-level mechanisms for a direct memory access. RawMemoryAccess represents a direct addressable physical memory. The content of this memory can be interpreted, for example, as byte, integer, or short (RawMemoryAccessFloat can be used for floating point numbers). The classes ImmortalPhysicalMemory and ScopedPhysicalMemory provide the possibility to allocate Java objects in the physical memory.

For practical use, the RTSJ has several powerful easy-to-use mechanisms. Other features such as the immortal memory are potentially dangerous and have to be used very carefully. As objects in the immortal memory never free their allocated memory (until the runtime is shutdown), continued or periodical object creation can easily lead to a situation where the system runs out of memory.

Combining the RTSJ with a real-time GC [5], as explained in [7], can overcome some limitations. In doing so, it is possible to access the heap from the real-time part without limitations and to directly synchronize real-time and non-real-time parts of an application. Unfortunately, the resulting application will not be executable on each RTSJ-compliant JVM.

Also, some conceptual problems of the RTSJ have to be noticed. While the RTSJ provides a defined API, some aspects of the behavior can depend on the underlying RTOS. The API is the same, but the semantics may change! This particularly applies to scheduling and the possibility of implementing scheduling strategies other than the primary scheduler.

4.2 Real-Time Core Extensions

The second consortium working on real-time specifications for Java is the JConsortium. One group within the JConsortium is the Real-Time Java Working Group (RTJWG). This working group has created the Real-Time Core Extensions (RTCE), finished in September 2000 [2].

The general goal of this specification can be characterized by reaching real-time behavior for Java applications with a performance regarding throughput and memory footprint comparable to compiled C++ code on conventional RTOS. Thus, this specification aims at providing a direct alternative to the existing real-time technologies. It was assumed that for typical applications in this field, the cooperation between the real-time part and the non-real-time part is limited.

As a result of these requirements, the general idea of the RTJWG was to define a set of standard real-time operating system services wrapped by a Java API (the “Core”). In doing so, the standard JVM was not changed but extended by a Real-Time Core Execution Engine. The components of the core are portable, can be dynamically loaded, and may be extended by profiles providing specialized functionalities.

4.3 Real-Time Data Access

Another working group within the JConsortium is the RTAWG. The focus of this group is the application field of industrial automation. Hence, the resulting “Real-Time Data Access”

(RTDA) specification focuses on an API for accessing I/O-data in typical industrial and embedded applications rather than on features supporting hard real-time requirements. Analyzing the typical requirements for the usage of Java in industrial applications, the general idea behind the RTDA is that the support of real time is a basic requirement, but that hard real time with sophisticated features is needed only in a few cases. More important is a concept for a common access to I/O data for real time and non-real-time parts of an application as well as the support of typical traditional procedures regarding configuration and event handling in this domain.

Following these conditions, the RTDA considers real-time capabilities as a prerequisite, assuming that the real-time and non-real-time applications run on a real-time-capable JVM.

4.3.1 Real-Time Aspects

The RTDA supports real time by using permanent objects and creating an execution context for these objects. Permanent objects are not garbage collected and have to implement the interface Permanent-Memory Interface, or the "permanent memory creation" has to be enabled by invoking the method enable() of the class PermanentMemory.

4.3.2 Event Handling and I/O-data Access

As the event handling and the I/O-access are the core components of the RTDA, it defines a dynamic execution model supporting asynchronous as well as synchronous access to I/O -data. The main instances responsible for creation of appropriate objects are the Event-Managers (asynchronous) and the Channel-Managers (synchronous). Every Event-Manager is created out of a DataAccessThreadGroup, which controls the priority of this Event-Manager. Depending on the type of event, there are different Event- Managers such as:

- IOInterruptEventManager (InteruptEvent)
- IOTimerEventManager (PeriodicEvent or OneShotEvent)
- IOSporadicEventManager (SporadicEvent)
- IOGenericEventManager (all types of events, more than one event)

Although there is no direct reference implementation for the RTDA, the Java package for SICOMP industrial-PCs by Siemens closely follows this specification of the J-Consortium. This particularly applies to the JFPC system (Java for process control) in relation to the I/O concept of the RTDA.[3, 8] The resulting structure of RTDA is given in Figure 2.

4.4 Comparison

After comparing all three specifications, it can be stated that all of them follow different approaches but also focus on different application areas. The RTSJ and RTCE have the goal to provide a more general way of enabling Java to become usable in the real-time domain, while the RTDA focuses on the special application field of industrial automation and understands real-time capabilities only as one of several requirements that are typical for this area.

The RTSJ is the most general approach, following the idea of "making Java more real-time." Therefore, the RTSJ can be used well by experienced Java programmers, as all enhancements are close to common Java concepts. Considering the number of products implementing the specification, currently the RTSJ has the highest acceptance. In contrast, the RTCE follows other premises, focusing on providing a performance comparable to the state-of-the-art (e.g., C++) solutions on commercial RTOS for hard-real-time requirements. The JVM was not changed, but extended by a separated "core," which realizes the real-time features. The RTCE provides real-time functionality following the typical concepts of today's real-time application development, and therefore it is assumed that it will be used by experienced real-time programmers. Hence, the RTCE is the first choice for problems dealing with hard-real-time constraints and high-performance requirements although, at the moment, the lack of a sufficient number of available products reduces the applicability.

The RTDA provides a complete concept for the application development in industrial control engineering. Real-time capabilities are seen only as a needed prerequisite for such applications, but the main part deals with the handling of I/O-data and interrupts. The concept is very similar to typical concepts used in conventional control systems and is therefore easy to understand for programmers working in that domain. Nevertheless, the RTDA is a very "closed" world and therefore it is not easy to adapt all ideas of using reusable components when programming control applications in Java according to the RTDA. Besides the JFPC system, which is available only for very powerful and expensive hardware (a Linux implementation will be developed and opens here new possibilities [12]), the RTDA also lacks a wide range of products.

Currently, it is still not clear if there ever will be the one specification, or if each of these specifications will find acceptance in a certain area.

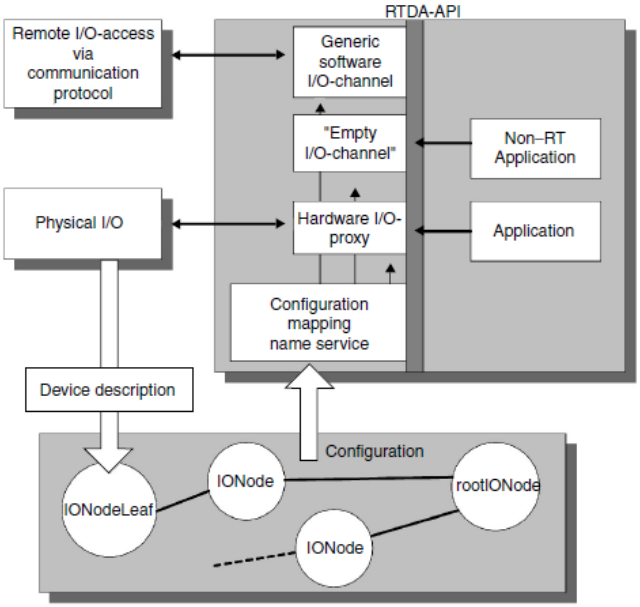

FIGURE 2. RTDA — overall architecture.

5. Java Real-Time Systems

To ensure the real-time capability of a Java application, independent from the used JVM, the underlying operating system (OS) has to be real-time capable as well. Therefore, in particular in the area of embedded systems, different possibilities of combining JVM and real-time OS exist. In general, there are four different categories:

1. Real-time JVM on conventional RTOS. In this case, a real-time JVM is running on a "normal" RTOS. This is typical for systems where the possibility to execute Java is an additional feature (of course, it can be the only feature that is used) and where the resources allow one to run a JVM with sufficient execution time. Normally, this is the case for systems such as IPCs and Soft PLCs, less for conventional PLCs because the operating systems and processors of conventional PLCs make the integration of a real-time JVM difficult. Examples of such systems are the SICOMP-IPC with JavaPackage [8] and the Jamaica-VM [7].
2. JVM and RTOS as one integrated system. If the only function of the RTOS is to run a JVM (this means the whole RT-application is written in Java), it can be useful to integrate both JVM and RTOS as one product and tailor the RTOS exactly to the needs of the JVM. Such systems can be a basis for full Java-PLCs.
3. Conventional RTOS with Java co-processor. If the resources of the system are limited with respect to the requirements of a JVM, it can be useful to have a special Java co-processor for the execution of the Java Byte-code. This results in an increased execution speed, as well as in an increased complexity of the system hardware and software.
4. Java processors. Finally, there are systems with a processor that executes Java Byte-code directly as native processor code. Here, the JVM is realized in "hardware" so that the conventional OS level can be dropped. This results in a very efficient (and therefore fast) Byte-code execution, but the flexibility of such systems is reduced because non-Java applications cannot be executed. The typical application for the last two combinations are intelligent I/O devices.

6. Control Programming in Java

Java RTS 2.0 was the first Java RTS product [17] release to include a real-time garbage collector (RTGC) that includes an enhanced version of an innovative real-time GC algorithm that provides the Java developers very fine-grained control over dynamic memory management in a real-time environment. The important point about the RTGC provided with Java RTS is that it is fully concurrent, and it can thus be preempted at any time. There is no need to run the RTGC at the highest priority, and there is no stop-the-world phase, during which all the application's threads are suspended during GC.

With some diligent care and guidance from faculty, any engineering technology student can quickly take advantage of Java RTS features such as strict thread priorities. This can be accomplished by simple substitutions in the java program code, though most motivated students will want to do more than that. The faculty here at DeVry University attempt to indulge students in such activities before they have even been exposed to control theory or DSP courses.

Appendix A-1 is an example of an instructor led program where the student edited, compiled and displayed the output. The purpose of this program is to show how general Java workhorse

discrete Fourier Transform and other control theory methods [16] can be introduced at an earliest stage to engineering technology students with the tools and concepts they will further reinforce in future courses.

The TestDFT application class given in Appendix A-2 uses class Fourier and invokes its methods.

Now the students are ready to implement an FFT as a Java method. It is defined in the Fourier class that also defines the discreteFT() method. See the Appendix A-3.

Here is a sample of TestDFT results:

```
f[0] = 0.0Xr[0] = -1.1275702593849246E-16  Xi[0] = 0.0
f[1] = 0.5Xr[1] = -4.5102810375396984E-17  Xi[1] = -6.505213034913027E-17
f[2] = 1.0Xr[2] = -5.898059818321144E-17  Xi[2] = -3.426078865054194E-17
f[3] = 1.5Xr[3] = 3.469446951953614E-17  Xi[3] = -1.5872719805187785E-16
```

In the next step, the students applied this method to compute the FFT on a 2Hz cosine wave. They were instructed to take 64 data samples over a 2-second sample period. The program first computes the FFT to obtain the frequency spectrum for a 2Hz cosine wave. Then the program below was used by students to perform an inverse Fourier transform that reconstructs the 2Hz cosine wave from its frequency spectrum. Next, we implement the FFT as a Java method. It is called the fastFT(). The fastFT method source code is shown in Appendix A-4.

Next thing to do is to test the fastFT() by applying it to the composite cosine signal that were processed earlier. The amplitude time history for a signal containing three different frequency components is generated and sent to fastFT () method. The TestFFT class source code is shown in Appendix A-5.

Here are the sample partial TestFFT results:

```
f[0] = 0.0 Xr[0] = 14.118483415068333 Xi[0] = 8.299076200364345
f[1] = 1.0 Xr[1] = -4.320681943456142 Xi[1] = -11.70913283509337
f[2] = 2.0 Xr[2] = 11.5931285252644 Xi[2] = 11.382484516157067
f[3] = 3.0 Xr[3] = 0.20410293867180673 Xi[3] = -9.83647671361791
…
f[60] = 60.0 Xr[60] = 0.012868248913439052 Xi[60] = 0.058867631665321246
f[61] = 61.0 Xr[61] = -0.20682636511072414 Xi[61] = 0.025288953333891355
f[62] = 62.0 Xr[62] = -0.03895429084677146 Xi[62] = -0.007474788441969488
f[63] = 63.0 Xr[63] = 0.02651390602928162 Xi[63] = 0.0
```

This introductory example demonstrates how students can be introduced to basic concepts of signal processing and, with diligent care, be further guided in a similar way to best practices for implementing real-time control systems, as well as how to use the RTSJ interfaces effectively in the design of real-time applications.

6.1 Requirements of Control Applications and New Possibilities in Java

Today, IEC 61131-compliant languages are typical for programming control applications. Although there are concepts for encapsulation of functionalities and data (e.g., function blocks provided in IEC61131- 5), the main advantage of Java, compared to these languages, is the possibility to use object-oriented features such as encapsulation and inheritance. Of course, networking abilities and stability also play an important role, but object orientation enables completely new ways for code reusability and increases the efficiency of application developments in control programming. As it is not efficient to implement the whole application from scratch for every new project, it is important to encapsulate functionalities in classes for reasons of reuse. Depending on the concrete device, by means of these classes (or interfaces), generic functions such as specific communication protocols or easy access to specific devices can be realized. These existing classes can (if necessary) be modified or extended and then be integrated into the application.

Hence, notable potentials result for industrial automation. Based on the platform independence of Java, parts of applications can be easily reused. Manufacturers of components like I/O-devices or intelligent actors/sensors can deliver classes (representing their components) together with the hardware. Instances of these classes can then be used in the concrete application and provide methods for using the device functionality, concealing the concrete hardware access. In doing so, the manufacturer can provide a clear simplification for the programmer and reduce the danger of improper use of his own hardware.

For complex applications, the concrete hardware access (I/O), communication functions, or other basic functions can be abstracted on different levels. Here, special tailored classes are imaginable, for instance the usage of IEC 61499 function blocks. A simple example for such a concept shall be explained now more in detail.

6.2 Structure of a Control Application in Java — an Example

The goal of the following is to give recommendations for the structure of an application to support efficient programming. This can be ensured on the one hand by as much device-independent as possible , and on the other, by using a consistent procedure for the handling of I/O-variables, independently if these variables represent local I/Os or remote I/O-devices, which are accessible over different communication protocols. The communication can be realized by conventional bus-systems (e.g., CANBus) or via Ethernet (e.g., Modbus/TCP or EtherNet/IP). How should an application for these requirements be structured? In general, it is advantageous to use several levels for the encapsulation of functionalities in Java classes (see Figure 4). To explain the structure, a simple example shall be used. The hardware structure of this example is shown in Figure 3. A conveyer system consisting of several belts and turntables is controlled by a PC-based Java-PLC. As I/Os, two Modbus/TCP-Ethernet couplers are used.

Objects of the lower level, called here communication level, realize the access to the concrete hardware. For local I/O access (e.g., memory access or GPIOs of a processor), these classes are device-specific. In contrast to this, for the communication with remote I/O -devices or other controls, generic classes (availability of appropriate hardware assumed) can be used on different platforms. Today, such Java classes are available for the Ethernet-based protocols Modbus/TCP

and EtherNet/IP. In our example, the Ethernet coupler (and its I/Os) are represented by two instances of a class Modbus. These objects ensure the read- and write-access to all I/Os of the appropriate coupler and make it possible to read and write the I/Os. These objects, basically realizing a communication, can be used by classes of the next level (I/O-level), which represents the logical I/O-variables (e.g., digital or analog inputs or outputs). Here, a mapping from physical variables (representing physical I/Os) to logical variables is realized. If this mapping is flexible by using configuration information, for example, stored in a configuration file or as an input from an IDE, the control application can be implemented device-independently. In the example, every input or output is represented by an object of the class DigitalIn or DigitalOut, which is linked with the help of a configuration file to the appropriate Modbus-objects.

On the third level, complex control functionalities (e.g., emergency-stop) as well as representations of parts of the plant control system (e.g., components such as conveyors, drives, or generically usable actors) can be encapsulated in classes. Objects on this level, as well as groups of I/O-variables on the I/O-level, typically run in their own real-time thread. In the example, there are objects for each element of the transport system as instances of the classes Conveyor and Turntable. These objects provide methods for controlling these elements (e.g., start( ), stop(), turnleft( ), turnright( ),…).

This structure simplifies the reusability of huge parts of the control application. Thus, on the highest level (application level), an application can be implemented in a more abstracted, function-related way.

6.3 Integration of Advanced Technologies

The use of Java for control programming at the field level shows advantages, especially in systems where Java is also used for non-real-time applications. This applies for visualization or remote access for specific functions like maintenance and code download/distribution. In this context, new technologies also in the area of industrial automation such as agent-systems or plug-and-participate technologies play an important role.

For implementing such systems, special attention has to be paid to restrictions regarding the mutual interactions of the control application running under real-time conditions and the non-real-time application. If the real-time Java platform provides support in this respect (such asthe software I/O proxy of the RTDA), this can be realized quite easily. Unfortunately, most real-time Java products and specifications do not follow the requirements of industrial control applications so, special attention has to be paid to these aspects. As a general rule, it can be said that time-critical parts of the application will run with a higher priority, while non-real-time parts have priorities below the GC and will be executed as normal Java applications and, of course, restrictions such as synchronous calls from real-time to non-real-time part have to be observed.

For this case, an architecture is reasonable which decouples the control part and allows a synchronization at certain states of the system. The loose coupling of both parts, allowing the access to the control only in exact defined states, can be implemented by a connection layer using, for example, a finite state machine. This allows one to load, parameterize, and start control applications.

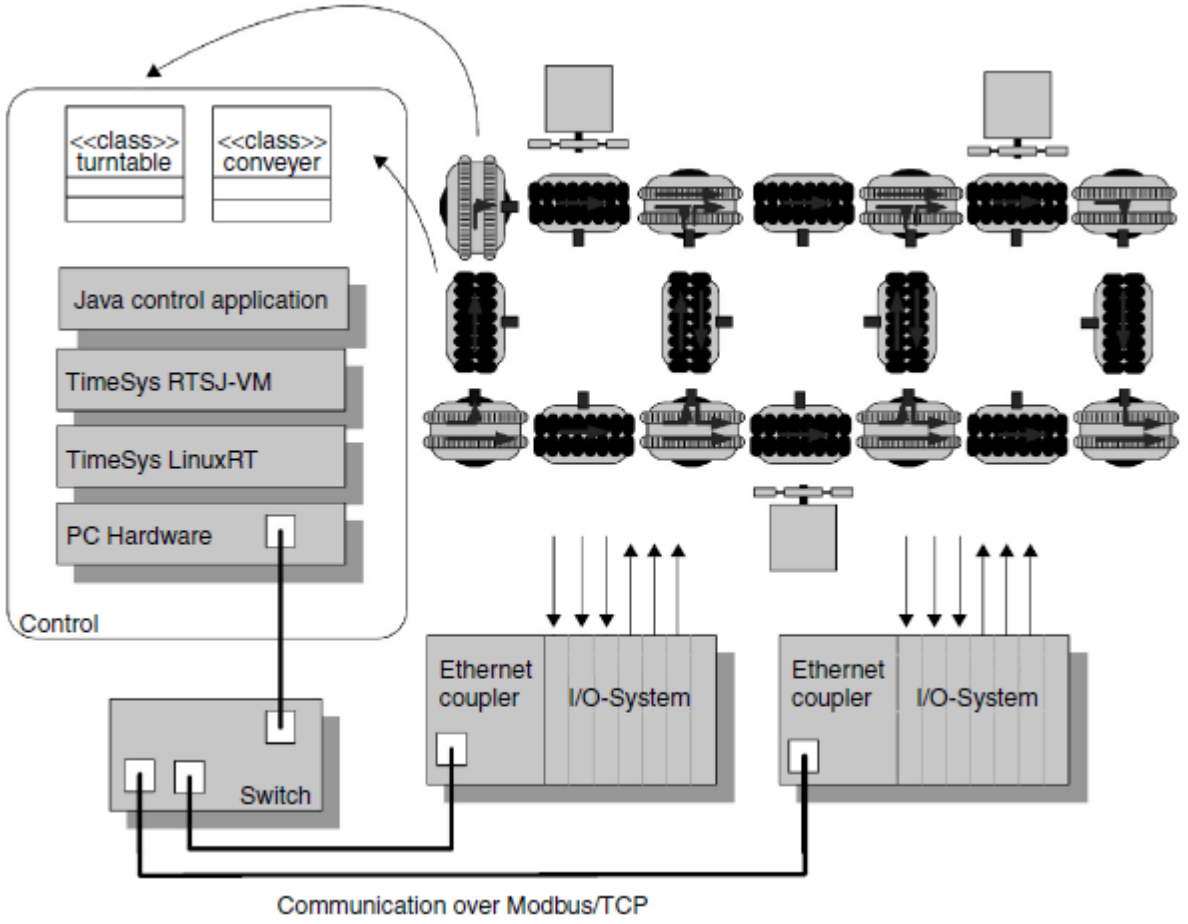

FIGURE 3. Hardware structure of the example.

As an example for such a system, the so-called “Co-operative Manufacturing Unit” (CMU) shall be mentioned. It was developed within the international research project PABADIS [11]. This project aims at creating a highly flexible structure for automated production systems, replacing parts of the traditional MES-layer by concepts using technologies like mobile agents and plug-and-participate technologies. The CMU is the entity in the system providing the functionalities of automated devices (e.g., welding, drilling) to the PABADIS-system. Although it is also possible to connect conventional controls (e.g., PLCs) to the system, the fully Java-based CMU is the most advanced concept. It avoids the additional communication effort from the object-oriented Java-world to IEC 61131-compliant languages and provides all the advantages of Java stated before. The outcomes of the PABADIS-project give an idea of the possibilities that technologies like Java can bring to future automation systems.

6.4 Migration Path for the Step from Conventional Programming to Java Programming

As mentioned before, it is necessary to provide a possibility to migrate in a certain way from the conventional IEC 61131-based programming to Java programming. This way can be opened by applying the ideas of IEC 61499 and the definition of special function blocks for the application

parts given in the component level, I/O level, and communication level of Figure 4. Based on these predefined structures and self-designed control application blocks programmed in IEC 61131-compliant dialects (which can be automatically translated to Java), a new way of programming can be established.

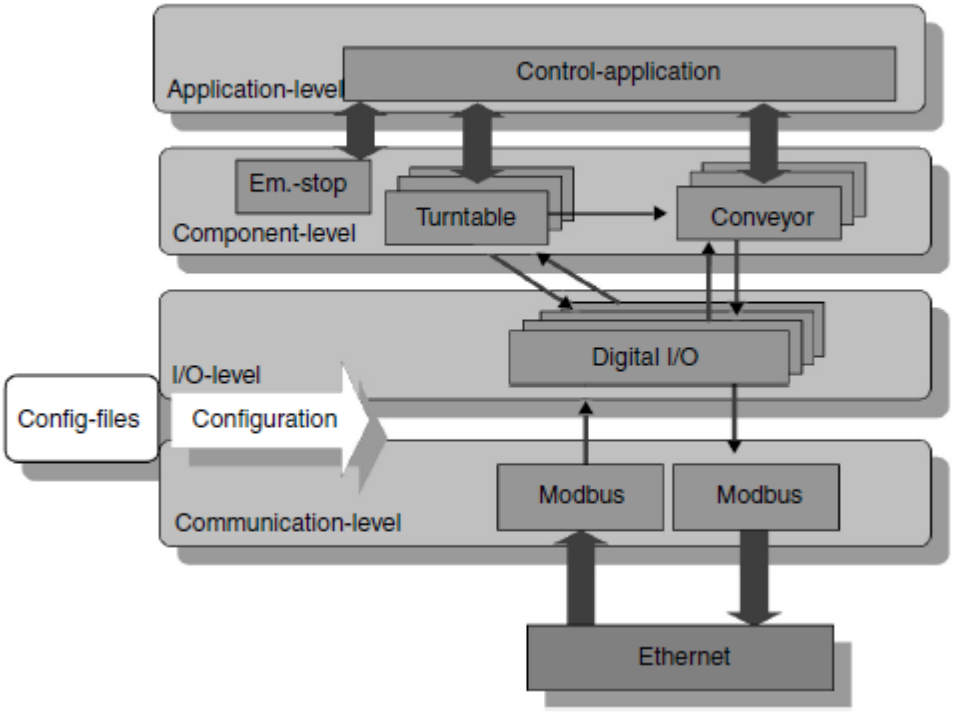

FIGURE 4. Structure for control applications in Java.

7. Conclusions

To summarize, it can be stated that Java technology has reached a status where the technical prerequisites regarding use in industrial automation even in the area of control applications on limited devices are fulfilled. There are Java2 Editions allowing an adaptation to the requirements of the platform, and several products using different approaches providing real-time capabilities in Java are now available. This allows for the use of Java on nearly all types of devices at the field level. A common standard for real-time Java is important against the background of retaining platform independence of Java also for real-time devices. Now the development of concepts regarding how the advantages of object-oriented, high-level languages can best be used for increasing the efficiency of application development (by supporting reusability and providing abstract views on the Java application) is momentous. Besides this, Java opens new possibilities for an easy integration of technologies like XML, Web-Services, (mobile) agents, and plug-and-participate technologies [11].

8. Bibliography

1. Shakib, J., Muqri, M., *A Taste of JAVA –Discrete and Fast Fourier Transforms,* American Society for Engineering Education, AC 2011- W241.
2. Shakib, J., Muqri, M., *Leveraging the Power of Java in the Enterprise*, American Society for Engineering Education, AC 2010-1701.
3. Real-Time Core Extensions, International JConsortium Specification, 1999, 2000.
4. Real-Time data Access, International JConsortium Specification 1.0, November 2001.
5. Dibble, P., *Real-Time Java Platform Programming*, Sun Microsystems Press, Prentice-Hall, June 2008.
6. Siebert, F., *Hard Real time Garbage Collection in Modern Object Oriented Programming Languages*, BoD GmbH, Norderstedt, 2002.
7. Pilsan, H. and R. Amann, *Echtzeit-Java in der Fertigungsautomation Tagungsband SPS/IPC/Drives 2002*, Hüthig Verlag, Heidelberg, 2002.
8. Siebert, F., Bringing the Full Power of Java Technology to Embedded Real time Applications, *Proceedings of the 2nd Mechatronic Systems International Conference*, Witherthur, Switzerland, October 2002.
9. Hartmann, W., Java-Echtzeit-Datenverarbeitung mit Real-Time Fdata Access, Java™ SPEJTRUM,3, 2001.
10. Brich, P., G. Hinsjen, and J.-H. Jrause, *Echtzeitprogrammierung in JAVA*, Publicis MCD Verlag, München und Erlangen, 2001.
11. Shipjowitz, V., D. Hardin, and G. Borella, The Future of Developing Applications with the Real-Time Specification for Java APIs, JavaOne Session, San Francisco, June 2001.
12. The PABADIS project homepage, www.pabadis.org, 2003.
13. Jleines, H., P. Wüstner, J. Settje, and J. Zwoll, Using Java for the access to industrial process periphery — a case study with JFPC (Java For Process Control), *IEEE Transactions on Nuclear Science*, 49, pp. 465–469, 2002.
14. TimeSys, Real-Time Specification for Java Reference Implementation, www.timesys.com, 2003.
15. Hardin, D., aJ-100: A Low-Power Java Processor, Presentation at the *Embedded Processor Forum*, June 2000, www.ajile.com/Documents/ajile-epf2000.pdf
16. Schwab, C. and J. Lorentz, *Ethernet & Factory, PRAXIS Profiline — Visions of Automation*, Vogel-Verlag, Wuerzburg, 2002.
17. Palmer G., Technical Java - Developing Scientific and Engineering Applications, Prentice Hall, 2003.
18. Java RTS Descriptive Documentation, *Java RTSReadme.html*, July 2008, http://download.oracle.com/javase/realtime/doc_2.1/release/JavaRTSReadme.html

Appendix A-1

```
public class Fourier {

      public static double[] discreteFT(double[]fdata, int N, boolean
fwd){
            double X[] = new double[2*N];
            double omega;
            int k, ki, kr, n;
            if (fwd){
                  omega = 2.0*Math.PI/N;
            } else {
                  omega = -2.0*Math.PI/N;
            }
      for(k=0; k<N; k++) {
            kr = 2*k;
            ki = 2*k + 1;
            X[kr] = 0.0;
            X[ki] = 0.0;
            for(n=0; n<N; ++n) {
                  X[kr] += fdata[2*n]*Math.cos(omega*n*k) +
                  fdata[2*n+1]*Math.sin(omega*n*k);
                  X[ki] += -fdata[2*n]*Math.sin(omega*n*k) +
                  fdata[2*n+1]*Math.cos(omega*n*k);
                    }
             }
      if ( fwd ) {
            for(k=0; k<N; ++k) {
                  X[2*k] /= N;
                  X[2*k + 1] /= N;
            }
      }
      return X;
      }
      }
```

Appendix A-2

```
//TestDFT Program
public class TestDFT {

 public static void main(String args[]) {
       int N = 64;
       double T = 2.0;
       double tn, fk;
       double fdata[] = new double[2*N];
       for(int i=0; i<N; ++i) {
             fdata[2*i] = Math.cos(4.0*Math.PI*i*T/N);
             fdata[2*i+1] = 0.0;
       }
 double X[] = Fourier.discreteFT(fdata, N, true);
 for(int k=0; k<N; ++k) {
     fk = k/T;
     System.out.println("f["+k+"] = "+fk+"Xr["+k+"] = "+X[2*k]+ "
Xi["+k+"] = "+X[2*k + 1]) ;
 }
 for (int i=0; i<N; ++i) {
       fdata[2*i] = 0.0;
       fdata[2*i+1] = 0.0;
       if (i == 4 || i == N-4 ) {
             fdata[2*i] = 0.5;
       }
 }
 double x[] = Fourier.discreteFT(fdata, N, false);
 System.out.println();
 for(int n=0; n<N; ++n) {
    tn = n*T/N;
    System.out.println("t["+n+"] = "+tn+"xr["+n+"] = "+x[2*n]+"
xi["+n+"]  = "+x[2*n + 1]);
    }
  }
}
```

Appendix A-3

```
public static void FFT(double[] fdata, int N, boolean fwd) {
            double omega, tempr, tempi, fscale;
            double xtemp, cosine, sine, xr, xi;
            int i, j, k, n, m, M;

            k=0;
            for(i=0; i<N-1; i++) {
                  if (i<k) {
                        tempr = fdata[2*i];
                        tempi = fdata[2*i + 1];
                        fdata[2*i] = fdata[2*j];
                        fdata[2*i + 1] = fdata[2*j + 1];
                        fdata[2*j] = tempr;
                        fdata[2*j + 1] = tempi;     }
            k = N/2;
            while (k <= j) {
                  j -= k;
                  k >>= 1; }
            j += k;                                 }
      if (fwd)
            fscale = 1.0;
      else
            fscale = -1.0;
      M = 2;
      while( M < 2*N ) {
            omega = fscale*2.0*Math.PI/M;
            sin = Math.sin(omega);
            cos = Math.cos(omega) - 1.0;
            xr = 1.0;
            xi = 0.0;
            for(m=0; m<M-1; m+=2) {
                  for(i=m; i<2*N; i+=M*2) {
                        j = i + m ;
                        tempr = xr*fdata[j] - xi*fdata[j+1];
                        tempi = xr*fdata[j+1] + xi*fdata[j];
                        fdata[j] = fdata[i] - tempr;
                        fdata[j+1] = fdata[i+1] - tempi;
                        fdata[i] += tempr;
                        fdata[i+1] += tempi; }
            xtemp = xr;
            xr = xr + xr*cos - xi*sin;
            xi = xi + xtemp*sin +xi*cos;
            }
      M *=2;
      }
      if( fwd ) {
            for(k=0; k<N; k++) {
                  fdata[2*k] /= N;
                  fdata[2*k + 1] /= N;        }     }     }     }
```

Appendix A-4

```
public class Fourier {
public static void fastFT(double[] fdata, int N, boolean fwd) {
      double omega, tempr, tempi, fscale;
      double xtemp, cos, sin, xr, xi;
      int i, j, k, n, m, M;
      k=0;
      for(i=0; i<N-1; i++) {
            if (i<j) {
                  tempr = fdata[2*i];
                  tempi = fdata[2*i + 1];
                  fdata[2*i] = fdata[2*j];
                  fdata[2*i + 1] = fdata[2*j + 1];
                  fdata[2*j] = tempr;
                  fdata[2*j + 1] = tempi;}
      k = N/2;
      while (k <= j) {
            j -= k;
            k >>= 1;  }
      j += k;          }
if (fwd) {
      fscale = 1.0;
  } else {
      fscale = -1.0;}
M = 2;
while( M < 2*N ) {
      omega = fscale*2.0*Math.PI/M;
      sin = Math.sin(omega);
      cos = Math.cos(omega) - 1.0;
      xr = 1.0;
      xi = 0.0;
      for(m=0; m<M-1; m+=2) {
            for(i=m; i<2*N; i+=M*2) {
                  tempr = xr*fdata[k] - xi*fdata[k+1];
                  tempi = xr*fdata[k+1] + xi*fdata[k];
                  fdata[k] = fdata[i] - tempr;
                  fdata[k+1] = fdata[i+1] - tempi;
                  fdata[i] += tempr;
                  fdata[i+1] += tempi;  }
      xtemp = xr;
      xr = xr + xr*cos - xi*sin;
      xi = xi + xtemp*sin +xi*cos;
      }
M *=2;
}
if( fwd ) {
      for(k=0; k<N; ++k) {
            fdata[2*k] /= N;
            fdata[2*k + 1] /= N;}      }
    return ;  }       }
```

Appendix A-5

```
public class TestFFT {
public static void main(String args[ ]) {
      int N = 64;
      double T = 1.0;
      double tn, fk;
      double fdata[ ] = new double[2*N];

      for(int i=0; i<N; ++i) {
            fdata[2*i] = Math.cos(8.0*Math.PI*i*T/N) +
Math.cos(14.0*Math.PI*i*T/N) +
    Math.cos(32.0*Math.PI*i*T/N);
            fdata[2*i+1] = 0.0;
            }
Fourier.fastFT(fdata, N, true);
System.out.println();
for(int k=0; k<N; ++k) {
      fk = k/T;
System.out.println(  "f["+k+"] = " + fk + " Xr["+k+"] = " + fdata[2*k]
+ " Xi["+k+"]

= " + fdata[2*k+1]);
               }
        }
}
```