The Modular TORERO IEC 61499 Engineering Platform – Eclipse in Automation

Christian Schwab, Marcus Tangermann, Arndt Lueder
Univ. of Magdeburg, Center Distributed Systems – CVS@IAF
Universitätsplatz 2
39106 Magdeburg / Germany
{christian.schwab, marcus.tangermann, arndt.lueder}@mb.uni-magdeburg.de

Abstract

The current situation in automation is characterised by an ongoing trend towards distributed systems based on intelligent devices, the application of web services and the usage of Ethernet as the main communication media. The most important standard to deal with Distributed Control Systems (DCSs) is the IEC 61499 whose industrial application is still at the beginning. Within this paper, a platform for DCSs based on the open source framework Eclipse will be presented. The platform implements the DCS’s methodology realised by specific tools in the form of plug-ins covering the whole life cycle of a control system, ranging from the engineering process to the maintenance process during runtime of the system. A core functionality of the engineering platform is the Function Block System Designer supporting the IEC 61499 standard.

1. Introduction

In the last decade, process and control systems have experienced a strong trend towards an increasing complexity, variability and flexibility which can be seen in the following facts:

- product spectra have got a bigger size,
- lot sizes became smaller,
- production systems are now more flexible, and
- used manpower has decreased [1].

To cope with the increasing complexity, variability and flexibility of process and control systems, two main aspects have to be considered.

On the one hand, the complex hardware structure will be based on small, intelligent, network-connected, and plug-and-play enabled control devices. These devices will form a community of sensors and actuators enabling the application of system design principles forming and applying mechatronical units without the need of central supervisor entities like Programmable Logic Controllers (PLCs). This improves the variability, flexibility, scalability of control systems, supports the management of the complexity by allowing concepts such as resource sharing, and enables capabilities such as self-

configuration, faulty device replacement scenarios etc. [2, 3]

On the other hand, new software approaches to control and manage a network of intelligent devices are needed. A layered software architecture provides an abstraction in different levels, supporting device independent programming of the control application, and allows the integration of different hardware platforms and legacy system components. [4, 5]

With respect to this, an important milestone to cope with distributed resources is the international standard IEC 61499 [6, 7] whose event-driven architecture defines a generic software architecture to deal with and to manage distributed Industrial Process Measurement and Control Systems (IPMCSs). Currently, in the academic world the Function Block Development Kit [8] and the CORFU framework [9] are supporting the IEC 61499 approach, but these tools are stand alone applications which do not manage all aspects necessary to cover the whole life cycle of an IPMCS. Within industry, the usage of IEC 61499 is not yet widespread, but several tools, e.g. PROFINET CBA (Component Based Automation) based iMAP from Siemens [10] and L-Force Engineer from the German Lenze [11], deal with distributed hardware and software resources based on intelligent mechatronical modules.

With respect to this, the research project TORERO [12, 13] (Total life cycle web-integrated control) – funded within the IST (Information Society Technologies) initiative of the European Commission – aims at specifying a Distributed Control System (DCS) applied in factory automation which is called TORERO system.

A TORERO system consists of a new type of Internet connected and Ethernet based field devices (TORERO Device, TD). Such a Java based device provides plug-and-play capabilities to increase the flexibility of the DCS with respect to (re-) configuration and maintenance and also to support the minimisation of down time in case of device replacement scenarios.

Based on this infrastructure and in connection with Software Servers and an Integrated Development Environment (TORERO IDE, TIDE), the total life cycle of the DCS, ranging from the engineering phase and
managing of the system during runtime, to the maintenance/re-configuration phase and termination of the system will be supported.

In the following, an overview of the architecture of the DCS specified in TORERO and an introduction to the IEC 61499 standard will be presented. Based on the DCS’s infrastructure, the methodology of the system and the implementation view of the system will be introduced.

For the implementation the open source framework Eclipse [14] has been chosen. Therefore, in the following, an introduction of Eclipse and subsequently the usage of this open source framework as the basis for the TORERO Integrated Development Environment (TIDE) will be presented.

2. TORERO DCS

2.1. TORERO Control Device

The smallest part of a Distributed Control System in TORERO is an intelligent control device. This device is called TORERO Device (TD) and is a mechatronic component that can interact in a TORERO environment. In collaboration of all TDs the required control functionality will be realised [13, 15, 16].

Basically, a TD is a sensing or actuating component equipped with suitable hardware and software. Beside the physical components, an operating system is built on top of the hardware containing the processor, RAM, storage, I/O connections to the sensor/actuator element, the Ethernet interface, etc. The hardware can never be accessed directly by the control application, but only by so-called device functions. A device function is allocated to a specific device and provides basic functionalities of the TD such as `getPositionValue`, `turnLeft`, or pre-processing of data, scale a value, etc. and is mounted directly on the operating system or on the Java Virtual Machine which is hosted by the TD. The device functions can be considered as an abstraction layer of the underlying hardware and allow for a hardware independent and generic programming of the control application. Each TD contains a Device and Parameter Manager which acts as an access point to the TD for the TIDE and provides features such as start/stop of the TD, managing of the system during runtime, to the maintenance/re-configuration phase and termination of the system will be supported.

In the following, an overview of the architecture of the DCS specified in TORERO and an introduction to the IEC 61499 standard will be presented. Based on the DCS’s infrastructure, the methodology of the system and the implementation view of the system will be introduced.

For the implementation the open source framework Eclipse [14] has been chosen. Therefore, in the following, an introduction of Eclipse and subsequently the usage of this open source framework as the basis for the TORERO Integrated Development Environment (TIDE) will be presented.

2.2. Methodology

The core of the methodology specified in TORERO is the engineering process which is covered by the following steps [4, 13, 19].

In the first step, the control application of the DCS will be designed. Therefore a set of pre-defined and/or self-developed IEC 61499 Function Blocks (see next chapter) will be connected independent of the underlying resources.

In parallel, all TORERO Devices within the network announce their presence and their connections by means of UPnP, the Universal Plug- and-Play mechanism. The announced network of TORERO Devices will be displayed in the TORERO IDE’s “Network View” [13].

In Step 3, the control application will be allocated automatically to the TORERO Devices depending on resource parameters of the devices and network capabilities. The output is an allocation table which will be processed further in the next step [20].

In Step 4, considering the allocation results, communication related code will be generated automatically by using object oriented principles based on the Strategy Pattern [21, 22, 23].

In the following steps, first the complete code – the control application and the communication related code – will be downloaded to the TORERO Devices using ftp mechanisms and second, the control application’s life cycle, ranging from start/stop functionalities to automatic update mechanisms via the Internet will be managed [24].

2.3. TORERO IDE

The implementation environment of the methodology as described above is the TORERO Integrated Development Environment (TIDE), which supports the user during set-up, runtime, maintenance, and termination of the system [13, 16, 25].

The platform runtime is the basic environment for the TORERO IDE. It provides basic functionality for user interaction, team support, a common object model as a basis for all further elements, and interfaces to databases. For storing information about connected devices and for storing program code and updates/bugfixes, the TIDE is
equipped with a database that is either integrated in the TIDE itself or can be accessed via a database interface. The TIDE user itself interacts with the TIDE via the workbench. This workbench provides all needed GUI elements like a graphical editor for connecting function blocks or a text editor with syntax highlighting for programming code.

The concept of the TIDE, which has been implemented on the basement of Eclipse, is strongly based on plug-ins, enabling powerful extensions of the TIDE. Therefore the plug-in mechanisms of the platform runtime will be used. All plug-ins are managed by the plug-in manager and loaded either at start-up or at runtime.

The TORERO IDE provides an interface to the TORERO Software Servers which can be accessed via the Internet. A TORERO Software Server is a basic source of software within the TORERO system and acts as a download server to support the user during the engineering and maintenance process.

The TORERO Software Server is strongly based on the application of web services using the Simple Object Access Protocol (SOAP [26, 27]) as the application layer within the OSI reference model.

3. IEC 61499 Standard

The IEC 61499 standard [6, 7], applied within TORERO, is based on a fundamental module, the Function Block (FB), which represents a functional unit of software, associated to a hardware resource of a control system. A FB instance is characterised by: its type name and instance name, sets of event inputs/outputs and data inputs/outputs, internal data, an Execution Control Chart (ECC) consisting of states, transitions and actions, which invokes the execution of control algorithms in response to event inputs, and a set of algorithms, associated with the ECC states. This structure is given in Figure 1.

An incoming event will be read by the ECC and can enforce the invoking of an algorithm in dependence of the active state of the ECC and the possible state changes enforced by the incoming event. When the execution of an algorithm is invoked, the needed input and internal data values are read and new values for output and internal data may be calculated. Furthermore, upon completion of execution of an algorithm, the execution control part generates zero or more event outputs as appropriate.

By properly connecting more than one FB, a distributed application can be defined. The event flow between ECCs of FBs determines the scheduling and execution of the FB algorithms and thereby the behaviour of the complete control application.

Within TORERO, different kinds of IEC 61499 based function blocks are used: Control Application Function Blocks (FBs), TORERO Proxy Blocks and TORERO Communication Blocks.

The Control Application FB contains the actual control application allocated to the TDs in Step 3 of the methodology.

A TORERO Proxy Block (TPB) is a FB that represents a device function in the TIDE during the design phase of the control application. Actual TD characteristics are hidden behind a TPB and thus TPBs contain neither alterable ECCs nor alterable control algorithms, while it is not allowed that they are freely distributed to other devices on the TD network.

A TORERO Communication Block (TCB) provides an interface to the communication network representing different bus protocols within the TIDE. Similar to TPBs, TCBs contain no alterable ECCs or control algorithms and cannot be distributed to other devices.

The Control Application FBs can be distributed among several devices with the restrictions described above. A device uses the causal relationship specified by the Control Application FBs to determine the appropriate responses to events, which may include communication and process events, utilising the different resources associated to the devices.

A resource is considered within the IEC 61499 standard as a logical subdivision within the structure of a device, which has independent control of its operations. The functions of a resource are to accept, process and return data and/or events to the process and/or communication interfaces, as specified by the applications utilising the resource, i.e. by the FBs associated to the resource.
4. Eclipse Framework

A common problem of today’s development environments and configuration tools are the huge differences in handling and applying each tool, varying between different vendors and resulting in user obfuscation, high training efforts, and high costs for customer support. From the vendors viewpoint, most new tools are developed from scratch with high implementation effort; re-use of existing components is often difficult or impossible.

Recognizing these problems at the end of the year 2001, well-known vendors of development tools, among others IBM, Rational Software and QNX, formed the initial eclipse.org board, an organisation aiming at creating a toolkit for an Integrated Development Environment, following the main vision: “Eclipse is an open (IDE) platform for anything, and for nothing in particular.”

Eclipse is open because its design allows an easy extension by third parties. It is an Integrated Development Environment (IDE) because it provides tooling to manage workspaces, to build, launch and debug applications, to share artefacts with a team, to manage code version, and to easily customise the programming experience. Eclipse is a platform because it is not a finished application but it is designed to be extended indefinitely. Eclipse is suitable for anything because it has been used successfully to build environments for wide-ranging topics. Finally, Eclipse has no particular focus on any vertical domain.

The software is available under the Common Public License (CPL). This license allows the development of commercial products from the Eclipse Platform, and in case of developing an open source extension it must be published under the CPL.

As stated above, the approach of Eclipse is to provide a platform for writing and executing tools, that seamlessly work together, so that users do not run into problems importing and exporting data between different applications and getting familiar with different user interfaces.

Software developers are able to write tools, which integrate themselves seamlessly into the framework by using well defined mechanisms and rules.

Therefore API’s, which consists of classes and methods are offered. New tools, which expand the functionalities of Eclipse, are called plug-ins while very complex tools are represented by many plug-ins. A plug-in may depend on other plug-ins.

Plug-ins are always coded in Java and typically consist of Java code in a JAR-library and a manifest file declaring its interconnections to other plug-ins. The interconnection model is simple: a plug-in declares any number of named extension points, and any number of extensions to one or more extension point in other plug-ins. An extension point may have a corresponding API, which is realized by Java interfaces. Other plug-ins contribute implementations of this interface via extensions to this extension point. Any plug-in is free to define new extension points and to provide a new API for other plug-ins to use. These declarations of extensions and extension points in the manifest are expressed by using XML.

The Eclipse Platform is fully implemented in Java but it uses precompiled code for native machines. And, in spite of Java’s reputation to be very slow at runtime, the Eclipse workbench is relatively fast. This is reached by using the Standard Widget Toolkit (SWT), instead of AWT or SWING. SWT is a platform dependent windowing toolkit, which is optimised for the current operating systems. Because the SWT is available for all important operating systems, applications which base on it will be platform independent. The most important advantage of using SWT is the increased speed of execution. If a widget is not supported by the underlying operating system, this widget will be emulated by SWT.

So it is possible to use all widget, and not only these ones, which are offered from all operating systems together, in comparison to AWT.

The major components of the Eclipse Platform, as shown in Figure 3, are the workbench, the workspace and the platform runtime. The extensible user interface (UI) is built around the workbench and depends on two toolkits: SWT and JFace. While SWT was introduced above, JFace is a further API built on SWT offering interfaces for simplifying common user interface programming tasks.

Figure 3. Structure of the Eclipse Framework

The Eclipse UI paradigm is based on editors, views and perspectives. Editors allow the user to open, change and save different files and objects. The downloadable Eclipse SDK (www.eclipse.org) includes a standard text editor, while other editors like XML editors or graphical ones, are supplied by other plug-ins. But existent editors may also be extended by plug-ins, for example a
C# IDE, realised as a plug-in, is allowed to extend the standard text editor by syntax highlighting.

Views are graphical elements for showing information about objects the user is working with in the workspace. For example, a To-Do list may show all errors which have to be adjusted in the actual editors object. More specific views are also supplied by other plug-ins.

Perspectives are combinations of different editors and views to represent a whole application implemented as a plug-in. Only one perspective is visible at a moment. But other editors and views can be opened additionally. A fast switching between different perspectives is also possible. Tools integrated into this editors-views-perspectives UI paradigm are well-defined ways.

Of course, within this paper only a short introduction to Eclipse and its “Ecosystem” could be given. For a detailed description please refer to [28, 29] and the Eclipse homepage.

In the following chapter, the application of the Eclipse to the TORERO project will be described.

5. Eclipse in Automation

Based on the flexible plug-in architecture of Eclipse, the structure for the TORERO IDE (TIDE) was developed as shown in Figure 4. Functionalities are derived from the steps of the control system design methodology described above and encapsulated in plug-ins allowing an easy exchangeability and extendibility of the components, whereas primary plug-ins provide basic functionalities (XML Importer, RDBMS, Plug and Play, and Device Model) and secondary plug-ins support all steps of the methodology (Function Block Editor, Compiler, JDT, Allocation, Web Services¹, Weaving¹, BootP¹, FTP¹, HTTP¹, Security¹). For an in-depth description of the TORERO IDE please refer to [25].

Several data sources are available that can provide information about the network of TORERO devices. These sources are:

- XML Importer, a plug-in that can import device description delivered by the vendor on a portable media,
- Plug-and-Play, a plug-in using UPnP for gathering information about devices, and
- RDBMS, a base plug-in which can read and write data from and to databases.

The data provided by the plug-ins described above are stored within the TORERO Device Model plug-in. This plug-in provides a representation of a TD within the TIDE and acts as a database for all actions performed by the TIDE during the engineering and maintenance process. It can be divided into two main parts:

- Device Model specific classes and
- Function Block specific classes.

The first group of classes contains device specific information such as:

- Basic information of the TD (vendor, serial number),
- Control hardware (processor type and vendor, storage size),
- Operating system (vendor version),
- Device Functions (OS based and Java based),
- Java Virtual Machine,
- Communication Interface of the TD (Ethernet interface),
- Physical device components (e.g. sensor elements), and
- Configuration data (e.g. IP address).

The Function Block specific classes describe all Function Blocks installed on the TD. In particular, these classes encapsulate the elements of IEC 61499 Function Blocks such as the algorithms, Execution Control Chart, and event and data connections.

The TIDE Device Model is automatically generated within the TIDE by extracting all relevant data from the TORERO Device Description when the TD is plugged-in.

Figure 4. Structure of the TORERO IDE

![Figure 4. Structure of the TORERO IDE](image)

Figure 5. Different Views of the Model

Additionally, the Device Model plug-in provides a call-back interface for other plug-ins, that provides information about

- the status of a device,

¹ Beyond the scope of this paper, not visible in Figure 4.
• devices going online/offline, and
• events triggered by devices.

Different views on the model show the strong capabilities of Eclipse as a framework for creating development environments. For example, the user of the TIDE can see data stored in the device model plug-in either in a simple tree view or in a more complex network view as shown in Figure 5.

This network view was created using the Graphical Editor Framework (GEF), a framework for rapidly creating chart editors within Eclipse. GEF is an excellent example how the Eclipse “Ecosystem” provides support for creating own applications rapidly. There is a wide variety of plug-ins available that can be integrated in an own project and helps shortening the development time. The most common license in the Eclipse world, the Common Public License (CPL), allows the commercial use of those plug-ins and the platform, so there are low restrictions of developing a commercial version of Eclipse based products.

As stated in the previous chapter, the control application within the TORERO architecture is developed using the IEC 61499 standard. At first, the Function Block Development Kit [8] was used as a function block editor started as an external application of the IDE. But due to some reasons – e.g. not very clean user interface, difficulties with the handling of the tool – and the missing integration into the framework, it was decided to develop an integrated Function Block Editor. Based on the experiences gained by the development of the network view, the Graphical Editor Framework was chosen as a base framework for the creation of the editor.

Having the modular structure of Eclipse and the TIDE in mind, it was decided to provide a generic editor with a clean Extension Points to be extended and used by other plug-ins. These Extension Points allows:

• **participation in the model** - changes within the model will be reported to interested plug-ins (adding blocks, event connectors, etc.);
• **compiling** - allows the participation of plug-ins in a translation process (e.g. transforming the blocks into Java).

![Figure 6. TIDE Platform with IEC 61499 Function Block System Designer](image)

The function block editor, called “Function Block System Designer” (FBSD), is open to any language for the implementation of the algorithms as well as the target language of the whole system. As an example implementation, the project consortium has chosen Java and proved the application of this language within a prototype demonstrator plant in industrial environment. Very helpful in this context is another plug-in provided by the Eclipse Platform: The Java Development Tools (JDT), probably the most known part of Eclipse, which allow an easy access to:

• structured model of classes (Compilation Unit) providing an abstract syntax tree (AST), and
• an API for generating Java code based on AST.

Applying the AST, an implementation could be provided that can work simultaneously on both the
model of the different function blocks as well as Java code.

Recognising the necessity to integrate the well known languages of the IEC 61131 standard [30], the authors are working on plug-ins that bring those language to Eclipse and to the FBSD.

Eclipse provides interfaces for the integration of languages to the framework. Currently, text editors for Structured Text and Instruction List are under development by the authors providing features such as

- syntax highlighting,
- code assistance, and
- code folding1.

Additionally, the work on parsers for those languages is in progress developing AST and APIs similar to the ones provided by the Java Development tools.

6. Synopsis and Outlook

Within this paper, an IDE Platform for Distributed Control Systems which is based on the open source framework Eclipse has been presented. The powerful plug-in concept and the usage of a variety of available plug-ins allows for a seamless integration of all tools in the form of plug-ins necessary to manage a DCS’s life cycle as follows:

1. Development of the control application,
2. Automation Network Creator using Plug-and-Play concepts (UPnP),
3. Consistent and uniform data storage used by all plug-ins,
4. Semi-automatic allocation of the control application to the underlying hardware,
5. Automatic generation of communication related code depending on the allocation,
6. Transfer of the control application to the devices using IT standards such as ftp,
7. Managing of the system such as start/stop during runtime, and
8. Provision of maintenance concepts incorporating web services and IT standards such as ftp, and SOAP.

With the Function Block System Designer which is under development at the moment, an easy to use tool supporting the IEC 61499 standard is introduced which is also seamlessly integrated in the platform concept and which allows the extension to other languages.

After the work on and with Eclipse, the authors are convinced that Eclipse is a very powerful framework for developing a generic platform for IDEs in the control area. To extend the development and to inform the Eclipse community about the project, the Eclipse foundation has been contacted regarding an official “Eclipse in Automation” project. Currently, a proposal is in preparations by the authors.

Within the proposal, the work from the TORERO project is adapted and brought into a more industrial like environment. It is considered to use a device model based on the Eclipse Modelling Framework (EMF), which can be used to create a more generic model. Several approaches for generating code from such generic models will be extended for code generation. Additionally, the integration of C/C++ into the platform is planned. Therefore, the already available C/C++ Development Tools (CDT) will be used. Also, in the future work and enabled by the plug-in concept and the clear interface definition, further tools in the form of plug-ins (Simulation, Debugging, Human Machine Interface) will be specified and implemented to obtain a reliable, powerful and fully integrated platform to manage a DCS.

Acknowledgement

The work presented within this paper was partially supported by the European Commission under the 5th Framework Program within the TORERO project (IST-2001-37573).

References


---

1 Allows to hide sections of source code while working in other parts of it.


