An IEC 61499 Based Approach for Distributed Batch Process Control

K. Thramboulidis1, Member, IEEE, S. Sierla2, N. Papakonstantinou1, K. Koskinen2 Member, IEEE,
1Electrical & Computer Engineering, University of Patras, Greece.
2Information Systems in Automation, Helsinki University of Technology, Finland.
e-mail: thrambo@ece.upatras.gr, ssierla@cc.hut.fi, npapakonsta@upnet.gr, kari.o.koskinen@hut.fi

Abstract—The Function Block (FB) construct has been adopted by recent IEC standards for the design of reusable, interoperable, distributed control applications. In this paper, an approach to exploit the benefits of this paradigm in batch process control is presented. A hybrid approach that integrates the FB model with the Unified Modeling Language is exploited and customized to the batch domain taking as starting point the industrially accepted SP88 standard. A toolset was customized to support the presented approach and demonstrate the applicability of the IEC61499 Function Block model in batch processing. Research experience with industrial engineers in the context of IEC 61499 and SP88 is used to motivate a development methodology that is sufficiently straightforward and efficient. The Java-based IEC61499-compliant run-time environment used for the execution of the control application is briefly described.

Index Terms—IEC61499, Function Block, batch processing, SP88, PFC, UML in control, FB run-time environment.

I. INTRODUCTION

To address the increased demand for a more flexible development process in the control and automation domain, the International Electro-technical Commission (IEC), has defined the basic concepts and a methodology for the design of modular, re-usable Distributed Control Systems (DCSs). The IEC61499 standard [1] defines the Function Block (FB) as the main building block and the way that FBs can be used to define robust, re-usable software components that constitute complex DCSs.

However, the IEC FB model does not fully exploit current trends in software engineering, such as object technology, model driven and component based development and does not define the way that FB design models will be implemented. A hybrid approach [2] that integrates the IEC61499 FB model with the UML overcomes these limitations. The Corfu framework (http://seg.ece.upatras.gr/corfu) and the Archimedes System Platform (http://seg.ece.upatras.gr/mim/archimedes.htm) provide the infrastructure required for the exploitation of the hybrid approach in the development process of re-configurable DCSs. The effectiveness of these platforms was demonstrated by various case studies [3][4].

In the context of this work the applicability of the IEC61499 function block model in the batch process industry is examined. A methodology for the application of the FB model to the batch process domain is proposed. The FESTO Mini Pulp Process (MPP), a laboratory system, is used to present a transformational approach that reduces the switching cost from the ISA SP88 [5] industrially accepted family of standards, to IEC61499. We are not aware of any other work that examines the applicability of the IEC61499 FB model in the batch process domain.

The remainder of this paper is organized as follows: In section 2, a brief introduction to the IEC61499 FB model is given, along with a reference to the hybrid approach that was adopted in the context of this work. The issues involved in introducing new standards to the batch domain are discussed from a practitioners’ perspective in section 3; these considerations are used to motivate our choice of research arrangements and laboratory system. In section 4, the proposed development process for batch control is presented. Section 5 describes the Java-based IEC61499-compliant run-time environment used to deploy and execute the application. Finally the paper is concluded in the last section.

II. THE IEC61499 FUNCTION BLOCK MODEL

A. IEC61499 basic constructs

The FB, the basic construct of IEC61499, consists of a head and a body, as shown in figure 1(a). The head is connected to the event flows and the body to the data flows, while the functionality of the function block is provided by means of algorithms, which process inputs and internal data and generate output data. The sequencing of algorithm invocations is defined in the FB type specification using a variant of statecharts called Execution Control Chart (ECC) [1].

Fig. 1. Graphical representation of Function Block type.

(a) Function Block type

(b) Execution Control Chart

An ECC consists of EC states, EC transitions and EC actions, as shown in fig. 1(b). An EC state may have zero or more associated EC actions, except from the initial state that shall have no associated EC actions. An EC action may have an associated algorithm and an event that will be issued after the execution of the algorithm. EC transitions are
directed links that represent the transition of the FB instance from one state to another. An EC transition has an associated Boolean expression that may contain event inputs, data inputs, and internal variables. As soon as this expression becomes true the EC transition fires.

FB instances are interconnected to form FB networks (FBNs), as shown in fig. 2. A FBN may be executed on a single device but it is usually executed on a network of interconnected devices that are equipped with the proper execution environment.

Fig. 2. The control application as a network of interconnected FB instances.

B. A hybrid UML based approach for the FB model

Undoubtedly the IEC61499 standard is a significant step to address the complexity of the next-generation control applications. However, many open issues have to be addressed by the standard in order to be considered by industry as a mature approach for the next generation agile manufacturing systems [6]. The “hybrid approach” [2] and the CORFU framework provide an effective development process and supporting toolset to address many open issues of the standard. This approach integrates UML with the already well accepted by control engineers FB construct, to cover the analysis and design phases of the development process. A model-driven approach is followed to move from analysis through design, to implementation. IBM’s Rational Rose is utilized to cover the analysis and early design phases. CORFU FBDK is used to automate the transformation of the so created UML models to FB design models. These design models are next translated to executable models following a fully automated model-to-model transformation process supported by the Archimedes system platform.

Any FB-based approach is not automatically accepted by professionals [7][8] and the broad range of design alternatives permitted by IEC 61499 [9] must be carefully restricted to obtain a development methodology that is domain-specific, straightforward and efficient. Such a methodology was successfully piloted with professionals in a one week laboratory experiment in June 2006, as described in section 3. This was the motivation for applying the hybrid approach to the batch processing domain.

III. THE BATCH DOMAIN

A. Introducing New Standards into the Batch Domain

The UML and IEC 61499 offer generic software development approaches that face competition from specialized proprietary products. Practitioners are unlikely to consider these new standards before support is provided for domains such as manufacturing, batch processes or continuous processes. Even if such support is present, practitioners are unlikely to be attracted by the best practices of software engineering, if business goals can be met more easily with less elegant methods [10]. It is possible that the industry as a whole will settle for inferior standards or proprietary solutions, resulting in serious losses for the entire community, if individual companies do not have an adequate incentive for standards adoption [11][12].

In this paper, a feasible migration path to the Corfu framework and the Archimedes System Platform is proposed for the batch industry. Such a path will be heavily influenced by the existing investments of a company in terms of knowledge, development tools and legacy software [13]; the cost benefit analysis of a company depends on the extent to which this investment can be utilized in the process of adopting a new standard. Much of this domain knowledge is embodied in the industrially accepted framework for batch process control defined in the ISA SP88 family of standards [5] and its IEC equivalent, IEC 61512 [14]. The leading commercial batch control systems, InBatch, VisualBatch and OpenBatch, are fairly compliant to these standards as far as models and terminology is concerned [15], so any SP88-based design patterns should be easily learnt and accepted by professionals.

This research focuses on applying the design principles incorporated in the SP88 standard to support the straightforward development of IEC 61499 based batch control applications. These standards define physical, procedural and control hierarchies that can be used to modularize the application; this task involves considerations that are not present in manufacturing applications, because control loops often cross the boundaries of physical modules [7]. The Procedure Function Chart (PFC) of ISA SP88 is taken as a starting point for application development, and the software design is heavily influenced by this standard. The proposed development guidelines should presumably be easily understood and accepted by industrial stakeholders, and this has been validated by an experiment.

B. Research Arrangements

The Festo Mini Pulp Process is a simplified laboratory version of the liquor circulation in pulp and paper processes (fig. 3). The liquor circulation was first described textually and then specified with the graphical PFC notation. The PFC specifies four levels: procedure, unit procedure, operation and phase. Each level is implemented with one or more constructs from the lower level. For example, a phase might open or close a valve and an operation might open a route from one tank to another involving several valves. A unit procedure contains the sequence for one part of the process such as filling a tank or cooking under certain temperature and pressure. The example procedure was composed of a sequence of five unit procedures: impregnation, black liquor fill, white liquor fill, cooking and discharge.

The impregnation unit procedure, which is shown in figure 4, is used to illustrate the application and the PFC notation; in a later section, the corresponding part of the control
application will be presented. The internals of the unit procedure are described as a sequence of operations such as EM2_OP1 and EM5_OP1. The parallel lines are split and join conditions that permit the concurrent execution of operations; the flow of control will stop at the lower set of lines until all operations have been completed. The first three operations in impregnation are responsible for opening a route from tank T200 to the digester T300 and for turning on a pump. Then, the execution pauses until the surface level inside T300 reaches the LS+300 sensor. Then the outlet of T300 is closed, the pressure increases inside the tank since the pump is still working. After a delay of Ti, the pump is turned off, so the pipe that was opened previously is closed. After these operations are completed, the digester is depressurized by opening the exit valve at the top for 2 seconds.

Fig. 3. The FESTO MPP Piping and instrumentation diagram (P&ID).

C. A Pilot Project

An IEC 61499 based control application for the Festo MPP was developed by teams of professionals and researchers during a one week event at the Helsinki University of Technology in June 2006. One goal for the event was to have team members produce compatible components that could be integrated in a straightforward way in order to construct the application. The second goal was to validate that our proposed engineering methodology is adequate, sufficiently easy to learn and efficient to apply. Previous research has shown that practitioners will not adhere to methodologies that are inflexible, time-consuming or prescriptive [16][17]. Therefore, the engineering methodology was presented as guidelines, which were a set of examples that illustrated PFC constructs and corresponding IEC 61499 designs. Participants were able to apply these guidelines successfully when they had the opportunity to occasionally resort to the help of more experienced team members working in the roles of project manager, system engineer and tester. These roles were assigned to invited participants who were given a two-day training before the event.

Due to the guidelines, designer productivity and teamwork was much more efficient than in previous events, and both academic and industrial participants stated that the clear guidelines kept them engaged in productive work instead of speculation on subtle design decisions. It was then decided that the experiment should be repeated using the CORFU ESS.

IV. THE DEVELOPMENT PROCESS

The objective of the proposed approach is to automate as much as possible the transformation of PFC models to IEC61499 FB models that can next be executed on FB compliant run-time environments. For this transformation process the hybrid approach is fully exploited.

A. Transforming PFCs to use cases

In a first step PFC diagrams are transformed to use cases [18] that are constructed using a commercially available UML CASE tool. Use cases by their definition are normally used for the specification of interactive systems. A use case, as defined in [19] is “a coherent unit of functionality provided by a classifier (system, subsystem, or class) as manifested by sequences of messages exchanged among the system and one or more outside users (represented as actors), together with actions performed by the system.”

Our first attempt to follow this definition and exploit our experience from applying use cases in continuous production systems such as the FESTO MPS [20], resulted in a rather false application of the use case concept in the Festo MPP. A use case was defined for each event of any PFC describing a unit procedure. The application of this approach resulted in three use cases for the impregnation unit procedure with the third one partially covering some of the functionality of the next unit procedure, i.e., the ‘Black liquor fill’. The so generated use cases represent functionality provided by the system and are described by means of messages exchanged among the system and one or more outside entities (valves, tanks, pumps, etc.) represented as actors. However, this approach does not result in modular reusable units of functionality. This was the motivation for
proposing a slightly different technique of using use cases in batch systems. According to this, we mainly focus on the specification of a unit of functionality or piece of behavior provided by the classifier and we loosely apply the other constraint that is based on the initiating object that should be an actor as stated in the use case definition; a use case “describes a complete sequence initiated by an object (as modeled by an actor) [19].

Our purpose in using use cases with batch systems is to represent, at this level of development, quantized pieces of functionality that is available to users but not necessarily initiated by them. A classifier can be used to initiate a use case. This classifier may be the previous use case but since this introduces dependencies between use cases, a specific class or subsystem that should capture the specific sequencing of use cases of the current configuration of the batch system is proposed to be used. This approach provides also more flexibility in the re-configuration process of the batch system.

The user has at its disposal chunks of functionality described in terms of use cases to configure and re-configure the batch system to meet its changing requirements. This provides the infrastructure that allows the control engineer to work for re-configuration purposes at the use case level. Figure 5 presents three such use cases that have resulted from applying this approach to the impregnation unit procedure of figure 4, to capture quantized pieces of functionality. Alternatively if such a low level of granularity in functionality is not required for the specific batch system or subsystem, a use case corresponding to each unit procedure can be defined. The “impregnation” use case shown in Fig. 6 is an example of such a higher level of granularity use case. However, if the low level use cases have already been described as shown in figure 5, the description of the impregnation use case will be quite different from the one shown in Fig. 6. It will simply define the order of execution of previously defined use cases. It should also be noted that only a simplified form of these use cases is given with focus on the mainline behavior ignoring alternate sequences, exceptional behavior and errors.

**Use case name: Fill the digester with impregnation liquor**
The system opens T200toT300 and T300toT200 paths. It next activates the pump of T200stoT300 path. When T300 Digester notifies the system (through LS+300) that it is full, the system closes the T300toT200 path as well as the T200toT300 path and the use case is terminated.

**Use case name: Pressurize digester**
The system opens the T200toT300 path and activates its pump. It resets the impregnation timer so the pressure in Digester is raised. When the impregnation timer expires the system closes the T200toT300 path and the use case is terminated.

**Use case name: Depressurize digester**
The system opens the T200toT300 path and sets the depressurization timer. When the depressurization timer expires the system closes the T200toT300 path and the use case is terminated.

**Use case name: Impregnation**
The system opens T200toT300 and T300toT200 paths to fill the digester with impregnation liquor from T200. When the Digester is filled the system closes the T300toT200 path and opens the T200toT300 to depressurize digester. When the depressurization timer expires the system closes the T200toT300 path and the use case is terminated.

For each use case, a first-level OID is initially created to capture the interactions with the mechanical process [2]. A detailed-OID is next constructed to capture the design decisions made at this phase. Fig. 7 presents the OID of the “filling of digester” use case. For the generation of this OID we assume the existence of one FB type for each mechanical unit involved in the specific use case. This FB will be constructed by the mechanical unit vendor to encapsulate the logic required to control the specific mechanical unit and provide an IEC61499-compliant interface for it. For example ValveFB is the FB type for valves and PumpFB the corresponding FB type for pumps. The IOD-Controller is defined to capture the control logic of the use case. To reduce the complexity of this FB type an intermediate level of coordination and synchronization hierarchy can be introduced. The T200toT300 controller FB type was defined to capture all the coordination logic of the mechanical units, i.e., valves and pumps that compose the pipe that connects the T200 impregnation liquor tank with T300 digester. This FB provides a higher level coordination between system’s components and simplifies the application development process by providing higher level commands for opening and closing the whole pipe. A problem that arises when applying this grouping is that some pipes such as the T200toT300 and the T400toT300 share the same valve i.e., valve V301. We decided to assign the valve to one controller (T200toT300) and provide an interface to it so as to allow the other controller (T400toT300) to request for the use of the valve. Another approach is to consider the valve V301 independent and give each controller the ability to directly access the valve. In this case mutual exclusion issues should be resolved appropriately. The approach adopted by the SP88 standard to address these issues, by decomposing the physical equipment hierarchy to units and equipment modules, does not provide obvious solutions to the above mentioned decisions.

OIDs are simplified if IEC61499 compliant mechanical units (industrial process terminators, IPTs) are used. In this case the IPT has embedded the corresponding FB type instance and there is no need to represent it in the OID. Corfu FBDK in its new version supports both types of IPTs giving more flexibility at the design level and a better integration of the application design specs with system’s layer design specs.
C. Moving to Function Block design specs

The resulting UML model of the control application is exported in XMI format that is supported by most of the UML case tools to support model interchange. These XMI models are imported into the Corfu FBDK and automatically transformed to FB network diagrams. Fig. 8 shows the automatically created FBN diagram for the “filling of digester” use case. FBNs are next merged to create the complete FBN diagram of the control application. The resulting FBN is very difficult to understand and verify, so the concept of leveled FBN proposed in [3] was adopted. “Using interface” as well as “implementation interface” as defined in [3] can be exploited to get a leveled FBN that greatly simplifies the development, testing and verification process of the design model.

Fig. 7. Object Interaction Diagram (OID) of the “filling of digester” use case (part).

FB type specifications and FBN specifications are saved in XML format that is accessible by Archimedes model transformers that allow the fully automated generation process of the executable model as well as its deployment on the target network of interconnected nodes. Our target execution environment for the Festo MPP case study is based on the DS80C400 embedded board. More information on the Festo MPP case study can be found at http://seg.ece.upatras.gr/seg/dev/FestoMPP.htm

V. A JAVA-BASED FUNCTION BLOCK EXECUTION ENVIRONMENT

To provide an execution environment for the IEC61499 FB model we selected the commercially available embedded board DS80C400-KIT that is based on the DS80C400 Network Microcontroller [22]. The DS80C400 network microcontroller offers the highest integration available in an 8051 device. It is equipped with a full application-accessible TCP IPv4/6 network stack that supports up to 32 simultaneous TCP connections and can transfer up to 5Mbps through the Ethernet MAC. Its maximum system-clock frequency of 75MHz results in a minimum instruction cycle time of 54ns. The 24-bit addressing scheme supports up to 16MB of contiguous memory. The board is equipped with the TINI platform [23], a microcontroller-based development platform with an extensible Java run-time environment that simplifies development of the network connected equipment.

Fig. 8. FB network diagram of the “filling of digester” use case.

The execution environment that is the non real time version of the one presented in [24] allows the deployment of distributed control applications over the Internet, so as to enable life-cycle management remotely and securely over a network of Java-based devices. It is composed of a set of classes that provide the infrastructure required by a Java based execution environment to support deployment and predictable execution of FB based design specifications as well as run-time re-configuration. Between the most important classes of this category are: DeploymentManagerEntity (DME), DataConnection Manager (DCM), and the Event-Connection Manager (ECM). The DME communicates through TCP/IP with the Engineering Support System (ESS) and provides to the ESS the functionality required to download FB types, create FB instances, and their connections, etc., to setup and initiate the execution of this part of the control application that have to be executed on the specific device. During the establishment of an event connection between two FB instances, the DME subscribes the consumer FB to the ECM by calling the subscribe() method. The application launcher can subscribe an FB instance to more than one output events that allows an FB to receive notifications from multiple event producer FB instances. Fig. 9 presents the most important components of the Java-AXE package and their interconnection in the development process of the executable control application.

The Java-AXE implementation Framework is a set of classes that are re-used by the FB2Java model interpreters to generate the executable model of the FB based design specifications. FBType, BasicFBType, ECC, ECState, InputEventMonitor, etc. are among the most important classes of this framework. During run-time when an FB instance reaches the point in its ECC where it should signal its output event, the fire() method, notifies the ECM for the specific event. The ECM, which is an active object (thread), notifies the FB instances, which have subscribed for the particular event, writing the event to their InputEventMonitors. The consumer FB instance is awakened, it reads from the InputEventMonitor the input events and enters its ECC in order to check if any of its transition conditions is satisfied.
VI. CONCLUSIONS

In this paper, an attempt to exploit the IEC61499 function block model in the batch control domain is described. A methodology based on the SP88 standard has been validated in an experiment with professionals, and subsequently this methodology has been incorporated to a tool-chain that automates most of the development process. The new approach exploits the best practices of software engineering in the form of UML and IEC 61499, and the goal has been to make this accessible to professional automation designers with minimal retraining. The software development approach presented here is a straightforward mapping from the PFC notation of SP88 to UML constructs, so designers are not confronted with the bewildering range of design options that must be considered when designing from scratch with UML or IEC 61499.

A laboratory system was utilized to demonstrate the applicability of the proposed approach. PFCs produced by control engineers were utilized to get the systems use cases that are next realized using OIDs. The obtained UML models are automatically transformed to FB design specs that are next realized using OIDs. The obtained UML models are automatically transformed to FB design specs that can be executed on the IEC61499-compliant Java Archimedes execution environment (Java-AXE package).

Even though the proposed transformational approach is not yet fully automated, it provides a promising way for exploiting the big investments in industry based on PFCs and SP88. The switching cost is considerably reduced because familiar concepts, terminology and design approaches from the batch domain remain useful in the context of the proposed approach. The contribution of this paper has been specifically focused on this aspect of the switching cost, since according to our previous experience with professionals, there is a tendency by many of them to reject the IEC 61499 standard simply because its learning curve is perceived as being very steep.

REFERENCES


Fig. 9. The process of developing and deploying the Java-based executable model from XML specifications of GB design models.