UML-based Development Process for IEC 61499 with Automatic Test-case Generation

Tanvir Hussain and Georg Frey
University of Kaiserslautern
Erwin-Schrödinger-Str. 12
D 67663 Kaiserslautern, Germany
{hussain,frey}@eit.uni-kl.de

Abstract

Development of distributed control applications within a standard framework was actually the strongest motivation behind the introduction of IEC 61499. Yet, scarcity of its application is due to the fact that the development is more complex as compared to the existing languages and standards. This paper addresses an UML-based development process for IEC 61499 which attempts to consider the aspects related to the utilization of the unique features that this standard brought into the field of industrial control applications. Reliability is an important consideration in this particular realm of software development and apart from formal verification tests are often performed to verify the software components. An automatic test-case generation methodology is introduced to complement the proposed development process.

1. Introduction

In the realm of Industrial Process Measurement and Control Systems (IPMCS) IEC 61499 [1] pioneers a standardized way of development of distributed software. Befitting the predecessor standard IEC 61131 [2] and up-to-date technologies (i.e., internet, WLAN etc.) which are manifested mainly by their potential of making distribution of hardware elements achievable and yet reliable to a certain extent, this standard renders the ability to handle the software-side complexities capitalizing on the trends from the world of business software.

The Object Orientation (OO) principally formed the basis for this new standard and hence the development process is also inspired by OO software development processes (i.e., design pattern and Unified Modeling Language (UML)). UML has a vast treasure of diagrams with well-defined semantic specifications. Apart from diagrammatic features it has textual elements like Object Constraint Language (OCL) which further augment its capability of specifying requirements and conceptual elements conceived at various stages of software development. In most of the cases not all of the diagrams are used, rather a chosen subset is used which suits the particular problem or problem domain.

This paper attempts to elucidate a proposed development process for IEC 61499 applications where selective diagrams of UML are used at various stages. In contrast to existing approaches its focus is principally on distribution of the specified software components on compatible hardware. This distribution should assure that specified functional as well as non-functional requirements are satisfied. Timeliness, often termed as non-functional requirement, is possibly the most important one to satisfy in most of the IPMCS software. Furthermore, automatic generation of test-cases is also proposed in this work. Tests can complement the practice of formal verification for checking the developed software. When ran on the actual target platforms they can furthermore check the appropriateness of a particular deployment.

The paper is structured as follows; in Section 2 the proposed development process is sketched along with its rationale and a brief comparison with other works in this realm. Section 3 discusses the work on automatic test-case generation. An application of the development process is shown in Section 4. Here, specific emphasis is put on automated generation of test-cases for unit/module tests of the IEC 61499 Function Blocks (FBs) and Resource entities. Concluding discussion and an outlook on future works is presented thereafter.

2. Development Process for IEC 61499

2.1. Motivation

IPMCS software is mostly real-time ones. With the inception of latest technologies concerning process interfaces (i.e., sensors and actuators) as well as communication infrastructures (i.e., fieldbuses, Ethernet and wireless networks) they are more prone to changes. These certainly necessitate a more structured approach in
concurrent engineering of mechanical, electronic and apply domain-specific modeling languages for the (MIM) paradigm is introduced and integrated in UML to detailed in [6] where the Model-Integrated Mechatronics Diagrams (CID) etc.. An extension of this concept is the newly introduced hierarchical Component Interaction mapping of elements of FB domain into UML is done extensive use of stereotype as well as a metamodel for software components of mechatronic systems.

One of the strengths of IEC 61499 is that it is assumed to be a tool for the development of distributed control applications. It is the structural hierarchy defined in the standard as well as the event based communicative interface among the FB constructs that account for this attribute. Moreover, the standard also allows dynamic deployment of the applications. An attractive utilization of these attributes of IEC 61499 could be accomplished if it is possible to achieve feasible and optimized dynamic deployment of the components of an application on available embedded devices. In terms of dynamic deployment it could then be possible to add additional functionalities, or even to change elements of control algorithms. This requires the specification to be accompanied by a proper set of timeliness and other constraints to be used in judging feasible deployment. Additionally, the smallest entity that can be deployed on an embedded device should also be chalked out during the development. The proposed UML based process is aimed at fulfilling these requirements.

2.2. Related works

At its inception the development of IEC 61499 applications concentrated principally on design pattern based developments [5]. Use of UML diagrams and corresponding artifacts began in the context of the work of CORFU Engineering Support System (ESS) development [3]. The result is a development process from requirement elicitation using UML diagrams (e.g., use cases and sequence diagrams) till implementation and a proposal of a layered architecture consisting of 4 layers hierarchically building up abstractions for fitting the structural hierarchies of IEC 61499 applications. The mapping of elements of FB domain into UML is done using – within others – the concepts of stereotypes and the newly introduced hierarchical Component Interaction Diagrams (CID) etc.. An extension of this concept is detailed in [6] where the Model-Integrated Mechatronics (MIM) paradigm is introduced and integrated in UML to apply domain-specific modeling languages for the concurrent engineering of mechanical, electronic and software components of mechatronic systems.

The UML-FB language proposed in [4] introduced extensive use of stereotype as well as a metamodel for the description of IEC 61499 constructs in terms of classes. In this context the behavioral modeling is proposed to be done using sequence or collaboration diagrams at inter-FB, resource and device level and using state diagrams at the intra-FB level. A tool is also developed in the course of the work that can convert UML entities drawn in Rational Rose UML tool to corresponding FB entities.

In [7] the use of UML is shown to design Functionality Based Controllers (FBC). State diagrams are used to model the ECC while class diagrams are used to show the dependencies and relations among the FBC units. The overall controller implementation then contains a triplet of Scheduler, Selector and Synchronizer to schedule the FBCs dynamically. This work is later augmented by automatic generation of FB types from UML representations [8].

In [10] the issue of specification and verification of IEC61499 applications is addressed where, apart from UML based specification of FBs, corresponding formal models are proposed. The formal models allow formal verification of the specified FBs through model checking. Testing can also be used where tests are supposed to be derived from the UML based specifications.

A recent survey on the use of UML during the development of IEC 61499 applications can be found in [9]. Another recent survey [11] focuses on the use of formal methods in developing IEC 61499 applications.

2.3. Proposed development process and its rationale

The proposed development process (cf. Figure 1) follows the popular V process model for software development [12]. It begins with requirement elicitation by means of use case diagrams and sequence diagrams, specifying particular scenarios. The constituent classifiers in the sequence diagram are either actors (user of the control program or the controlled process itself) or independent functional units which obviously will have process or communication interfaces. Therefore, objects representing these independent functional units can be mapped to IEC 61499 resources. The messages specified in the sequence diagrams portraying the usage scenarios will then reveal the interface requirements.

The requirement specification – especially the sequence diagram – is then analyzed to get the resource entities of the IEC 61499. This step is done automatically such that a component diagram is generated where the components correspond to IEC 61499 resources. The communication interfaces can be either publisher-subscriber instances or other kind of Service Interface Function Blocks (SIFBs). At this point of development it is not specified whether these SIFBs realize local or distributed multicast. This particular design option is kept open till the deployment is decided.
Figure 1. Proposed process model for development of IEC 61499 software.

The next step in the development process is concerned with elements of finer granularities namely, the FBs within the IEC 61499 resources. The SIFBs that should realize the process and/or communication interfaces are already specified in the previous step and therefore the remaining constituent FBs are mainly those corresponding to the realization of control algorithms. These FBs can be user-defined as well as off-the-shelf ones. Naturally, classes are the modeling counterpart of FBs. To build a clear and unambiguous mapping the classifiers needed for proper class to FB mapping have to be defined. The metamodel defining the basic FBs is shown in Figure 2. In this metamodel the interface and the body of a FB is differentiated at the top. The interface then consists of event and data input and output port objects each of which is further associated with a certain event or data type. The data input and output port classes are abstract since data ports should be of specific type (i.e., Boolean, integer, float etc.). The input and output event ports are further associated respectively to data input and output ports using the “WITH” relation in the IEC 61499 FB specifications.

Considering the body separately renders the opportunity to include variations regarding the scheduling of the constituent actions of the ECCs [13]. Moreover, definition of the interface elements as separate types may allow defining strategies for the event and data receptions and consequent handling of those [14]. This is done through polymorphic definition of concerned operations in the classes. A particular choice of execution scheduling and/or event and data handling has major influence in model-based analyses (i.e., schedulability, resource usage, load forecasting etc.). Therefore this choice has to be included in the UML based specifications.

Now the behavioral models of the resources are defined as well as the behavioral model of the FBs. Considering the reactive nature of the software component UML state chart diagram is an option for specifying the behavioral model. The state diagram based specification can further be augmented by activity diagrams which give the opportunity to model control as well as object flow through the indivisible algorithmic elements in a program [15]. Thus the individual behaviors of the FBs can be modeled using state diagrams and those among the FBs within or beyond the resource boundary using activity diagrams. An exclusive one-to-one relation can be established between the states of a state chart diagram and that of a corresponding ECC as well as between the signals and objects in the activity diagrams and the event and data interfaces of the FBs. Examples of these are provided during the case study (Section 4). It should be noted that the activity diagram based modeling concept does not include the description of executive actions that go on inside the SIFBs, a possible option of its inclusion could be through the use of interaction overview diagram [18].

Figure 2. Metamodel of basic FBs for class to FB type conversion.

With the above mentioned sequence of steps finally complete models of the IEC 61499 resources required for a particular application and the constituent FB elements are designed. Obviously what follows that is the automatic conversion of the UML models into appropriate FB counterparts, namely the components to resources, the stereotyped classes to FBs and the state diagram describing the behavior of individual FBs to corresponding ECC. The interconnections among the FBs are then formed using the swimlaned activity diagram that models the overall behavior. Following the transformation the IEC 61499 resources and its constituents are achieved.
The contents of the algorithms should now be written and using the code generator available in FBDK [19] it is then possible to generate an implementation in Java. Finally, the implementation should be verified or tested for certain criteria.

For easy application of the above mentioned process model a plug-in to the UML tool Visual Paradigm [20] has been developed. The developed plug-in is portable and can be activated on a Visual Paradigm installation when properly placed in the installation directory structure. This plug-in performs the automatic generation of the component diagram from the scenario specifications in terms of sequence diagrams. While designing the contents of the component, the user can import the FB libraries which will then generate corresponding class representations. Apart from building user-defined classes the user can then use FB classes from libraries. The user defined classes can also be converted to corresponding FB type definitions (.fbt files). During conversion the plug-in looks for activity diagrams defining ECCs as sub-diagram and if found will convert them to corresponding ECC.

The UML standard suggests the use of XMI for such conversions so that a tool can be used seamlessly for all UML environments that can import and/or export XMIs following the standard definition. Since not all tools support XMI and those that do use different XMI formats, direct access to the UML elements through API interfaces has been used. The API conceives the UML entities as objects. Therefore, it is possible to later integrate an XMI based conversion process without much effort.

3. Automatic Generation of Test-cases

Test is an important as well as costly step in the development process of software. In the realm of IPMCS formal verification is also a popular practice that can complement or substitute the need for testing but will require additional effort building up the formal model as well as huge computational power while realizing it. From the point of view of IEC 61499 test can have additional benefits. For example, when deployment of the applications is concerned, tests can help in determining the feasibility of a particular hardware element or platform in executing the application satisfying certain constraints especially those concerned with timeliness.

Tests are usually thought of covering certain aspects of an application. For example unit tests consider only particular programmatic modules, integration tests attempt to justify the integration of the modules and system tests focus on the complete system its functional and non-functional requirements and its target environment [21]. Use of UML during the development of OO-software renders a structured collection of specifications at various levels of abstraction. Therefore, it came out to be beneficial to use the UML artifacts for generating the test-cases whether manual or automatic. It can also be noted that the UML-based specifications can be used in both black box and white box variant of tests. UML also boasts a testing profile which defines how black box test-cases can be generated for component or system level tests. In the context of the presented work component level white box tests are considered.

During the above mentioned process of IEC 61499 compliant software development UML state diagram and activity diagram based behavioral models have been used. Using the state diagram representation it is possible to design test-cases following certain strategies. Among those round-trip path coverage is chosen. Round-trip path coverage is defined through the coverage of at least every defined sequence of transitions that begin and end in the same state [22]. In basic FBs the initial state can be considered to be the state where the round trips should commence and converge. A round-trip coverage of the state diagram will then reveal the missing event/action pairs [22]. The test-cases should contain a precondition that changes the values of the variables required for a particular transition to be actuated, an event or sequence of events that are needed for the transition trigger and postconditions namely the output event that follows a state’s algorithmic executions. For generation of these test-cases a transition tree is generated from each of the state diagrams [23]. Following a transition tree a set of test-cases is then listed for a particular state diagram.

The activity diagrams provide similar information as control flow graphs though at a higher level of abstraction as compared to the source code based traditional control flow graphs. In the context of IEC 61499 applications activity diagrams representing single basic FB units can be used for test-case generation. It is also possible to generate test-cases considering the resource entity consisting of FBs as a single unit. Here the outcomes are test scenarios that cover a particular path and for forcing the path traversal certain input events should be actuated apart from setting certain input or internal data to their specific values.

For respective test-case generations from the activity diagrams control flow graphs are formed and searching through the graphs the scenarios and constituent cases are listed. The test-cases or scenarios are often linked to one another since a certain state of an internal variable can be reached only after execution of a particular scenario. In case of one-to-one dependence of such cases the generated test-cases are listed in sequence. Moreover, the test-cases regarding satisfaction of time constraints are also required. This is done manually assuming the use of a hardware emulator for injecting the hardware and network specific delays on the path execution.
4. Case Study

In this section, the applicability of the above mentioned development process along with the generation of the test-cases will be shown for a practical control application. The development of a control application for a module of a didactic modular prototype plant is considered. A schematic of the module is shown in Figure 3. For control the actuators marked with A’s followed by an index and available sensor inputs marked with S’s followed by an index are used.

Figure 3. Studied machine module.

The discussed machine module should operate in three different modes namely, normal operation, emergency stop and manual cleanup. During normal operation the cylindrical object should be drilled followed by a test operation where the drilled hole will be checked and then the object will be kept ready for the following machine module which will either store it if the test result is positive or otherwise will dispose it. The emergency stop mode is for stopping the machine module at exigencies and during the manual clean up mode the machine moves to a specific state so that the objects can be removed and normal operation can be started after the removal of the objects.

Figure 4 shows the use case diagram for the controller. For the development process discussed use case diagram or its textual representation plays the role of recognizing weak classification of the development task. For example, the operation modes recognized at the use case level are later more descriptively specified using sequence diagrams and the most important of those namely the one specifying normal mode of operation is shown in Figure 5.

Figure 4. UML use case diagram for the operations of the studied machine module.

Figure 5. UML sequence diagram specifying normal mode of operation.

Apart from the message exchanges among the actors and the recognized objects stereotyped as <<FBRes>> implying correspondence with IEC 61499 resources the sequence diagram also contains time-constraints specified using OCL. While specifying the time-constraints the instant at which an event leaves an object is termed $sendTime$, the instant at which an event arrives at an object is termed $arriveTime$ while the instant at which the event is attended (corresponding ECC state transition, if available, is triggered) is termed $receiveTime$ [24].

Considering the communication shown through messages in sequence diagram a component diagram is then generated which is shown in Figure 6. Outside the Control Software component border lies the actors namely the SensorBoard and ActuatorBoard. The ports and interfaces within the software should be realized through SIFBs. The communication between <<FBRes>> components will require subscriber and publisher SIFBs for required and offered interfaces while those to components outside the Control Software component will require specific user-defined SIFBs.

Figure 6. Component diagram generated automatically from the sequence diagram of Figure 5.
The constituent classes of the component are then developed. Apart from the SIFB classes the classes required for the components for this case are the classes containing control algorithms and for the drill motor controller named DMController a timer class that will time drill motors action. Since the tool can load classes corresponding to the FBS available in a particular library it is also possible to reuse available or once designed classes anytime later.

The behaviors of the basic FBS are then specified through state diagrams. As an example the state diagram for the main controller FB of the RTMController resource is shown in Figure 7. The corresponding algorithmic action is shown using the label entry and the output event using the label exit.

Activity diagrams are then sketched for each of the resources. That will reveal the control and data flow among the FBS of a resource. Figure 8 shows the activity diagram corresponding to the RTMController resource. The object flows need not to be shown explicitly. This is due to the fact that the sources and sinks of the data in this particular case have a direct correspondence with the associated events. If this were not the case object flows were needed to be shown explicitly. The swimlanes are allotted for the required FBS. For the sake of conciseness the controller for testing module is not considered in the diagram as well as in the next ones.

The transformation process now generates the corresponding FBS and their interconnections. The generated FB network is shown in Figure 9. Certain connections relating to the initializations are missing in the generated FB network this is due to the fact that the SIFBs are modelled explicitly. In case composite function blocks are required to be created to take advantage of component reuse, it have to specified in the structural specification namely in the class diagrams.

At this point it can be conceived that the individual modules namely the FBS as well as the resources need to be tested. As discussed in section 3 the test-cases are then generated. For the round-trip path coverage of the state diagrams transition trees are built from the diagrams and using these transition trees the test-cases are listed. A concise glimpse of the table for the state diagram depicted in Figure 7 is shown in Table 1. This table does not list all of the cases but only the important ones, the removed cases are those which test the else conditions repeatedly. Following the generation of control flow graphs from the activity diagram similar tables are also generated where scenarios and cases for coverage of the control flow are listed.
Apart from functionality tests, there could be tests which can determine either the feasibility of a software deployment on hardware platform or measure the performance of it. This appears to be an important factor when hard real-time systems like the one described here is considered [25]. This is also a practically relevant problem as highlighted in the technical documentation of ISaGRAF [26] – the only commercially available IEC 61499 tools – “Delays are added in the communication interface and in the algorithms execution that must be taken into account when designing such a distributed application”. For example, both the timeliness constraints specified in the sequence diagram of Figure 5 are more or less dependent on the hardware where the implementation will be run. The feasibility of the software component for a particular hardware should in such cases be either tested running recursive tests or done using static code analysis. In the former case the outcome will be statistical data which can later be analyzed for feasibility with respect to certain criteria and in the latter case calculated worst case values.

In what follows the conceived idea of testing the non-functional requirements like time constraints is elucidated. Figure 10 shows a manually designed sequence diagram that depicts a test scenario for testing a particular time constrained action as specified in Figure 5. The role of the hardware emulator here is to include the necessary delays calculated in terms of worst cases and the process simulator provides the inputs to the emulator. The <<VerdictAction>> nodes are allotted for the verdict of pass or failure of the test-case executions. The process simulator also maintains the timers to keep track of whether the time constraint is satisfied or not. This particular instance considers the resource as a unit. Therefore it is possible to apply and evaluate the effect of a particular scheduling and event handling policy assumed by the resource for its constituents.

5. Conclusions and Outlook

Compared to its predecessor IEC 61499 introduces more flexibility, more leeway for the developers with respect to the hardware and communication infrastructures, and above all it ushered in a standardized way of developing distributed control applications for the IPMCS domain. While the above mentioned features brought it close to the business software domain easing up the integration as well as use of a rich treasure of tools and expertise that domain had developed over the years, the development of the applications became more complex in comparison to the existing alternatives. Use of UML and related techniques is certainly a way to reduce this complexity and to utilize the advantageous features that this standard introduces. In the context of the presented work a process model for the development of IEC 61499 compliant distributed application has been proposed. The proposed approach uses UML based specification from the very beginning of the development process and goes on further till verification of the implementations through UML specification based test-case generation.

For the UML based development it is often necessary to define metamodels when conversion from a domain specific concept to UML or vice versa is required. Therefore, within the work a metamodel has been developed for a mapping between the IEC 61499 terminologies and the UML ones. While developing the metamodel certain factors have been considered which add certain flexible and advantageous aspects in developing IEC 61499 applications (i.e., the flexibilities in dealing with differing scheduling algorithms for controlling the execution of the FB algorithms, the leeway in integrating different event handling schemes).
Moreover, for the verification of the implementations automatic test-case generation has also been proposed. In the context of the work path testing of the IEC 61499 modules has been considered. In future, integration testing as well as system testing will be integrated into the process which will complete the development life-cycle according to the V process model. Moreover, the above mentioned factors considered exclusively in the metamodel had not been included into the implementations since for generation of the codes FBDK [19] has been used. Whereas it could be important to have generation of model compliant code such that the choice of event handling and/or scheduling is implemented in the code. This is foreseen in the future works. Furthermore, there should be the scope of evaluating the feasibility of deployment of IEC 61499 components on heterogeneous hardware platforms and on dynamic basis. Apart from dynamic tests static code analysis with the help of hardware specific information could therefore be useful in the evaluation process.

References