A UML-based Approach for Modeling Industrial Control Applications *

João M. Fernandes, Ricardo J. Machado and Henrique D. Santos
Dep. Informática, Universidade do Minho, 4700-320 Braga, Portugal
Telephone: +351-53-604454, Fax: +351-53-604471, Email: miguel@di.uminho.pt

Abstract:
The main purpose of the poster is to present how the Unified Modeling Language (UML) can be used for diagnosing and optimizing real industrial production systems. By using a car radios production line as a case study, the poster shows the modeling process that can be followed during the analysis phase of complex control applications. In order to guarantee the continuity mapping of the models, the authors propose some guidelines to transform the use cases diagrams into a single object diagram, which is the main diagram for the next phases of the development.

Introduction:
For the development of embedded systems, the authors propose a new process model, which is based on the following characteristics: (1) Operational approach; (2) Refinement and transformation of the specifications; (3) Spiral model; (4) Reverse engineering; (5) Automatic code generation for prototyping.

The main views for specifying the system are captured by the following UML diagrams: Use case diagrams are used to capture the functional aspects of the system as viewed by its users; Object diagrams show the static configuration of the system, and the relations among the objects that constitute the system; Sequence diagrams present scenarios of typical interactions among the objects that constitute the system or that interact with it; Class diagrams store the information of ready-made components that can be used to build systems and specify the inheritance and hierarchical relationships among them; and State-chart diagrams are used to specify the dynamic behavior of some classes.

The information that is represented in state-charts, objects diagram, and classes diagram is transformed into Oblog, which is a UML-based object-oriented modeling language that allows the system to be simulated and has automatic code generation capabilities. The Oblog environment generates sequence diagrams, as a simulation output, that can be compared with those previously created to specify the system behavior in order to validate the system's requirements.

The process:
The first diagram to be built is the context diagram of the system, that shows which actors interact with the system. It defines the boundaries of the system. The actors are anything that interacts with the system, but do not belong themselves to it.

The next task was to define the use cases of the system. The authors propose an extension to UML by adding a new tagged value to use cases, that was designated *reference*. Each use case can have a reference that follows a numbering scheme similar to the traditional DFD numbering. A top-level use case is assigned a reference (example: ref=9), and if this use case is eventually refined by other sub-use cases, they will have a reference that uses the super-use case as a prefix (example: ref=9.2). This numbering scheme can be repeated to any depth and it helps those involved in the project to relate all use case diagrams and will be also used during the transition from use cases to objects to ease the mapping between both models.

The most important and complex use cases can be refined. This allows more detail to be added to the initial use case diagram and also the project to follow a risk-driven process, where the most important or complex functionalities of the system are first tackled. After identifying all the use cases of the system, the next step is to describe their behavior.

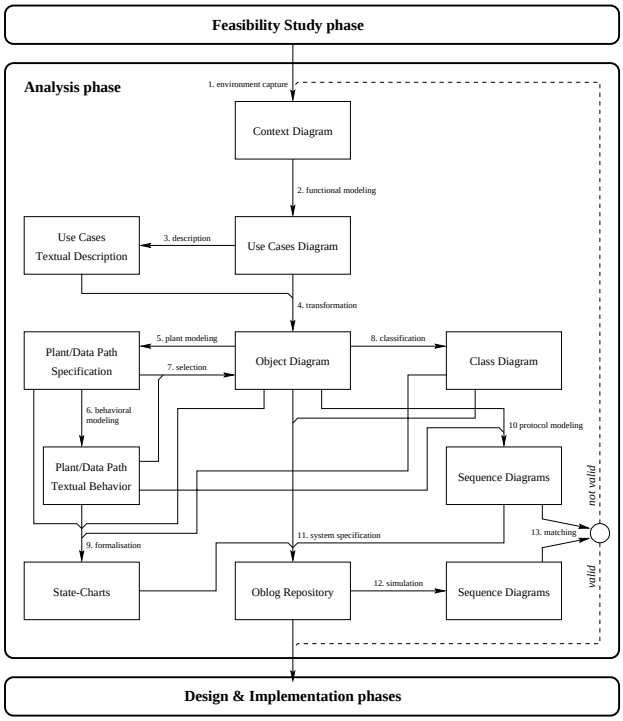

Figure 1: The process model for the project.

Object diagrams are important to show the components that constitute the system. Transforming the use cases that divide the system in a functional way into objects is a critical task, that demands some creativity from developer, since usually there is no direct mapping from use cases to objects. Several use cases can give origin to one single object, a single use case can give rise to a couple of objects, and eventually there are intersections among uses cases and objects.

The authors present a systematic strategy, based on the object types (interface, entity and control) pre-

*This work has been partially funded by the Portuguese Science & Technology Foundation project PRAXIS/P/EEI/10155/1998, *Reconfigurable Embedded Systems: Development Methodologies for Real-Time Applications.*

sented by Jacobson et al., for finding the objects of a given system based on its use cases that consists of a 4-step rule set:

step 1: Transform each use case in three objects (one interface object, one data object, and one control object). Each object receives the same reference as that of its corresponding use case appended with a suffix that indicates the object's category (*i* for interface, *d* for data, *c* for control).

step 2: Based on the textual description, for each use case it must be decided which of the three object categories must be maintained to fully represent, in computational terms, the use case, taking into account the whole system and not considering each use case *per si*, as in a reducionistic approach.

step 3: The survival objects must be aggregated whenever there is mutual accordance for a unified representation of those objects.

step 4: The obtained aggregates must be linked to specify the associations amongst the existing objects.

This approach aims to obtain an holistic set of objects (resources of the system), so that the inter-relations amongst the objectified use cases can be successfully simplified in order to obtain a reduced number of relevant and pertinent system-level objects.

As a user-readability pursuit, it is always a good practice to encapsulate as much as possible the system's representations using packaging. Each package defines a decomposition region that contains several tightly semantically-connected objects. The packages should be further specified, in the next project development phases, in what concerns its architectural structure.

The majority of the OO methodologies do not pay too much attention to the object diagram. Usually, the class diagram is built firstly, but in this project the order was reversed. To develop embedded control systems, the authors believe that it is more important to have a good object model than a good class diagram, because the elements that do constitute the system are the objects and not their classes. This was the main reason to first identify the objects and to later classify them (to select the classes to which they belong). We are not advocating to not work out the class diagram or even to ignore it (the best situation is having good object diagrams and good class diagrams). Instead, we promote that the focus should be directed towards the construction of the object diagram.

During the classification of objects the class structure is built, modified, or ideally just used. Reuse can be achieved in three different ways, during the classes discovery. First, if there are more than one object of the same class, their definition is specified in just one place. Second, if classes with similar properties are found, hierarchical relations among those classes can be defined. Finally, when the class of an object is described, it is possible that the developer recognizes the existence of that class in a library, which allows it to be immediately reused.

The class diagram is usually viwed as a template for a set of applications that can be obtained from it. In other words, the class diagram is a high-level generalization of the system. Whenever the developers define the way classes are interrelated, they are indicating all the systems that can be obtained from those classes.

With this perspective, it is common, in several methodologies, to not build the object diagram, since it automatically results from the class diagram. Whenever an object diagram is constructed, it is necessary to guarantee that the relations expressed in the class diagram between two classes also exist between instances of those classes. This is the main reason that methodologies usually impose (or suggest) class diagrams to be elaborated first than object diagrams. There is an additional task in which it must be assured that there is consistency between the information that is described by both diagrams. This fact can be interpreted as a symptom that some information is being unnecessarily replicated. We see the class diagram as a repository of previously defined objects' specifications ("a raw material store"), that can be used to develop any application.

For those objects that have a complex or interesting dynamic behavior, a state diagram should be specified. Object 9.3.c is responsible for controlling the movements of the radios along a Hidro line. Due to its great complexity, this object was decomposed into smaller objects, each one responsible for coordinating one node (a set of plant resources). Since the different nodes of a single Hidro line have not the same configuration, each kind of node requires a different state-chart, although there are similarities among them. For instance the state-chart of an upper node with 3 lines is similar to an upper node with 3 lines and one elevator that transports palets to the lower level. A class hierarchy can best indicate the relations amongst those different categories of node controllers.

Conclusions:

The poster presents how UML can be used in real industrial projects to model embedded control systems. The authors consider UML as an adequate language for industrial projects, since it is intuitive for non-technical people, it covers the main views of a system, it is independent of the platform and it is a standard.

The approach presented in the poster puts more emphasis on objects rather than on classes, which is one of its main divergences in relation to the traditional object-oriented approaches. Thus, the transformation from use cases into objects is one of the most important tasks within the development process. An holistic approach is followed during this transformation, so that it may be possible to obtain the object diagram that best maps the user's requirements into the system's requirements.

To ease the mapping of the models, tagged values (designated reference) are associated to the modeling elements (namely, use cases and objects). This mechanism is considered to be very useful, since it allows to circum-navigate throughout the whole complementary views of the system model, enabling the implementation of an operational approach within the spiral model-based analysis phase of system development.

Since some objects of the system present dynamic behavior, state-charts were used because they are adequate for modeling that view of the systems.

This UML-based modeling approach was validated with a real industrial case study. Although this paper just covers the analysis phase, the authors are using UML models in the design and implementation phases. The authors are aware that the approach needs to be applied to more projects in order to gain more experience and to improve some of the guidelines, namely in what concerns real-time constraints and model refinements during the design phase.