Abstract

To develop a CBTC system with safety and reliability, it is essential to develop CBTC software correctly. Model-driven development (MDD) is one of the approaches pursuing the correctness of software. With the assistance of software engineering technologies, MDD is realized nowadays.

In this case we adopted the Harmony, which is an MDD process from Telelogic. According to the Harmony process, we categorized the requirements into use cases and assigned them to development cycles. For each development cycle, we conduct analysis, design, implementation, and testing. Because the Harmony is a kind of spiral model, the model grows bigger as we complete each cycle.

As for software modeling, we modeled the CBTC software based on UML notation. The CASE tool generated the source code from the model easily, which enabled the fast iteration for each development cycle. Using the animation feature of CASE tool, we could verify and validate the complex functional requirements.

In conclusion, the MDD approach, which is supported by CASE tools, showed good applicability in developing the CBTC software with complex behavioral characteristics.

1 Introduction

To reduce the cost of infrastructure facilities and maximize the carrying capacity of railways, communication-based train control (CBTC) system is attracting the notice of many railway stakeholders. CBTC systems consist of real-time embedded systems with high level of safety and reliability. In developing the software for CBTC systems, we should apply various software engineering techniques to comply with the standards, such as IEC 62279, related to safety and reliability of railway software.

Recent advances in software engineering, especially in computer-aided software engineering (CASE) tools, create a new trend called Model-driven Development (MDD). MDD refers to a range of development approaches that are based on the use of software modeling as a primary form of expression. Code can be generated from the models, ranging from system skeletons to completely deployable products. In addition, we can verify the models by executing (simulating) them on a computer, and find out the errors we’ve made during the analysis and design phase.

So, if we apply the MDD approach properly, we can benefit from the advantages of MDD. The objective of this paper is to present our attempt to apply an MDD approach to the development of our CBTC system. To achieve this aim, we adopted a development process, called Harmony, and a CASE tool, called Rhapsody.

2 Model-Driven Development

A model is a coherent set of formal elements describing something (for example, a system, bank, phone, or train) built for some purpose that is amenable to a particular form of analysis. When a model’s subject matter has a high degree of abstraction, the model is closer to the eventual user’s language – that is, a smaller gap exists between a non-computer expert and the model [4].

Engineers have continued to raise programming abstraction levels. Today’s object-oriented languages let programmers tackle problems of a complexity they never dreamed of in the early days of programming. Model-driven development (MDD) is a natural continuation of this trend. Instead of requiring developers to spell out every detail of a system’s implementation using a programming
language, it lets them model what functionality is needed and what overall architecture the system should have [1].

MDD's defining characteristic is that software development’s primary focus and products are models rather than computer programs. The major advantage of this is that we express models using concepts that are much less bound to the underlying implementation technology and are much closer to the problem domain relative to most popular programming languages. This makes the models easier to specify, understand, and maintain [5].

**Harmony Process**

Harmony is a model-driven hybrid iterative process, supporting top-down systems engineering as well as incremental software engineering. It enables seamless transition from systems engineering to software engineering by using the UML.SysML as paradigm independent modeling language. It is supported by Rhapsody tool, maintaining common database for both the systems engineering phases and the software engineering phases [3]. The general form of the Harmony process is shown in Figure 1.

![Harmony Hybrid Spiral Process](image)

**Figure 1: General Harmony Hybrid-spiral**

The Harmony process is a general systems-development process that, while emphasizing the real-time and embedded-software development aspects, includes the steps to produce general-purpose software and systems. Harmony is a highly scalable “medium-weight” process, striking a balance between static heavyweight processes and lightweight, so-called “agile methods” such as Extreme Programming, while incorporating aspects of both [2].
3 CBTC Development

3.1 System Engineering Workflows

The systems engineering portion of the Harmony process consists of three phases: Requirements Analysis, System Functional Analysis, Architectural Design. Figure 2. shows overall sequence and the artifacts of each phase.

Requirements analysis starts with the analysis of the process inputs. Customer requirements are translated into a set of requirements that define what the system must do (functional requirements) and how well it must perform (quality of service requirements). Once the requirements are sufficiently understood, they are clustered in use cases. A use case describes a specific operational aspect of the system. It specifies the behavior as perceived by the user and the message flow between the users and the use case.

Figure 2: Systems engineering phases of the Harmony Hybrid-spiral

Figure 3: ATP Trackside Use Case Diagram
Figure 3. shows the use case diagram of ATP trackside system. “Manage Train Information” use case is about managing the train running number and the status of the train. “Manage Position Report” use case handles the position reports of trains so as to acquire position information of individual train and informs the occupancy of each interlocking area to IXL. “Manage Movement Authority” use case makes the movement authority and send it to the train, considering the safe rear end of the preceding train and route. It also deals with the reverse movement, working area, temporary speed restriction functions. Emergency situation is treated in “Manage Emergency” use case. For the case of broken rail detection or bad weather conditions, “Manage Detectors” use case is prepared. Lastly, “Manage Communication Session” use case manages the wireless communication between the trackside and the train.

Focus of the system functional analysis phase is the translation of the functional requirements into a coherent description of system functions. During this phase, system operations are identified by analyzing the interactions between the system and the actors, considering the system as a black-box.

Figure 4: Black-Box Use Case Scenario

Figure 4. shows a black-box use case scenario. The scenario is described in sequence diagram notation, considering the system as a black-box and representing the interactions between the system and the actors. In this case, the situation is for “establishing communication session”. At first, the ATP onboard system sends the message “reqProcessCommunicationInit()” to the ATP trackside system. After the ATP trackside system receives this message, it makes the version information message ( structureSystemVersionInformation() ) and sends it back to the ATP onboard ( reqProcessSystemVersionInformation() ). If the ATP onboard receives the version information, it tells the trackside that the session is established ( reqProcessSessionEstablishedReport() ). Then the ATP trackside registers the train in its database ( registerTrainIntoDB() ) and reports to the ATS that the communication session is established ( reqProcessSessionEstablishedInformation() ).

In architectural design phase, the identified system operations are allocated to each subsystem and the interfaces between subsystems are defined. By revealing the subsystems, black-box analysis is converted to white-box analysis.

3.2 Incremental Development Workflows

The Incremental Development Cycle(IDC) is an incremental process, in which the requirements elements are selected from the previously specified requirements set (represented on one hand as a UML/SysML model and on the other as a set of test vectors) and vertical slices of the system are
constructed and validated. IDC consists of five phases: Analysis, Design, Implementation, Testing, Increment Review. Figure 5. shows the structure of IDC.

![Incremental Development Cycle (IDC)](image)

**Figure 5: Incremental Development Cycle (IDC)**

Analysis is all about “what” and not “how”. An analysis model is focused on the identification of the properties and characteristics required of any acceptable solution. During the analysis phase, we identify objects and classes with object identification strategies for each use case,. There are many ways to identify the objects and one of them is “Apply scenarios”. According to this strategy, we walk through scenarios and missing objects will become apparent when required actions cannot be achieved with existing objects and relations.

![ATEP Trackside Elaborated Sequence Diagram](image)

**Figure 6: ATP Trackside Elaborated Sequence Diagram**

Figure 6. shows an elaborated sequence diagram resulting from “Apply scenarios” approach. It is based on black-box use case scenario. This time, the “black-box”, the system, is expanded in detail, describing the internal elements and their interactions. In this diagram, “ManagePositionReport” represents the system. The dotted lines at the right of the system are internal objects identified, whereas the slashed lines at the left represent the actors. One arrow from an outside actor is expanded to several internal method calls. By expanding like this, we can get the responsibilities of each internal element.
Object analysis identifies the object roles inherent in the requirements set as well as their types, classes, attributes, behaviors, and relations. This is normally done by adding a class diagram that represents the collaboration of the objects to realize that use case.

Figure 7: ATP Trackside Class Diagram

Figure 7. shows a part of the class diagram of ATP trackside system. At the center of the diagram, there is “ATPTracksideController” class. It contains the “RadioInterface”, “Map”, and “Train” objects as its parts. In the case of “Train” objects, they are created and added in the container during run-time, when the ATP trackside newly registered a train after establishing communication session. A “Train” object can have “MovementAuthority”, “ReversingArea”, and “WorkingArea” objects if necessary. The “Map” object contains “InterlockingArea” and “PlatformArea” objects. When an unconditional emergency stop occurs, the case is added in the container as an “UCEmergencyStop” object.

If analysis is all about the essential properties of a system, then design is all about optimization of those properties. A good design is one that optimizes the important design criteria at the expense of the design criteria of lesser importance. In the Harmony process, there are 3 levels of design abstraction:

- Architectural Design: Identification of strategic design decisions that optimize the entire system at once
- Mechanistic Design: Identification of medium-level design decisions that optimize a specific collaboration realizing a single use case.
- Detailed Design: Optimization of individual objects with the application of “design idions”

The Implementation phase focuses on producing the high-quality executable units(components and subsystems) to be integrated into the prototype build. Integration is very straightforward, particularly with a high-quality model compiler such as exists within Rhapsody. Most of the time, virtually all of the code is generated from the tool by running the model compiler. Once the unit is built, it is subjected to unit-level testing. Init testing is a set of white-box tests that ensure that the component is internally free from defects.

Testing is, or should be, about validating the quality of an already high-quality system. Quality is something that is designed in and is a result of continual testing and debugging through constant execution of the evolving model. In the Harmony process, the primary set of operational and QoS test vectors are created as sequence diagrams by the systems engineers during the systems engineering activities(for system-level testing) or during the elaboration of the sequence diagrams during subsequence analysis and design activities.
4 Conclusion

Traditional approaches to develop a system have been working well in various domains, but they have limitations also. As the system grows bigger, it becomes more and more difficult to maintain consistency of models and artifacts. Moreover, in case of a system with complex behavioral characteristics, it is not easy to test and validate the functionality because the elements of the system interact together and the state of the system changes real-time.

Up to now, we have been conducting the first iteration of IDC and we will continue the spiral sequences for another year. In our attempt to apply Model-Driven Development approach in developing CBTC software, we could model the CBTC system consistently and test the system effectively with the assistance of a CASE tool. In contrast to traditional approach, we could make the working prototypes early and check the validity of model by simulating it quickly, thereby ascertaining the advantages of MDD approach.

References