Abstract—this paper presents principles for model based testing of embedded systems with use of critical knowledge about system and their environment. For critical knowledge representation UML models can be used. These properties have different priority, impact, dependency, which creates different situations for testing. Critical properties mentioned in this article can serve as input for generating test cases. V-model, as a well known model, provides another approach for representing critical properties of software architecture and testing embedded systems. This model is often used in medical projects, e.g. radiotherapy. Some principles are taken from radiotherapy software project in Siemens Healthcare. Tracing of critical properties in V-model is also part of this article.

Keywords—critical knowledge, model based testing, embedded systems, UML

I. MOTIVATION

The progress in technological solutions allows designing, implementing and using systems in various spheres of everyday life. Except of information and control systems are becoming popular various embedded systems working in specialized hardware architectures, with specialized operating systems and using specialized peripherals developed for a specific application domain. As an example of these systems used for medical purposes can be radiotherapy.

These kind of embedded medical systems are developed by top-down method. V-model is a frequently used model for these cases.

![V-model diagram]

Figure 1. V-model

The main parts of this model (Figure 1.) are creating requirements of developed system, analysis, detailed design, implementation and multiple types of testing. Each type of testing on the right side of V-model is binded with a specific phase of software life cycle on the left side of a model (e.g. unit tests are created based on design specification and integration tests are created based on requirement specification).

First step of this implementation type is the analysis of requirements with the goal to identify critical properties of final system. Those are the properties, which have high impact on the success of a system. These are especially outer properties such as safety, reliability, user-friendliness (in case of interactive systems) and adapting communication protocols of embedded systems with communication protocols of peripherals or technological devices, where communication is required.

Critical properties are regarding to architecture of final system, ways of accessing required processes, data and control flows. All these identified critical properties represented with convenient models together with acquired relevant experiences of managers, analytics, designers, programmers and testers are creating critical knowledge, which usage in software life cycle is crucial. One area of using this knowledge is testing of systems.

To identify critical requirements of embedded system in whole model, special keys are introduced. These keys accomplish one basic property – traceability. It is assured that the requirement is fully implemented and tested from the beginning to the end of V-model. The critical requirement key is traced from upper-level documents to lower-level documents, e.g. key from System requirement specification (SRS) is traced to Subsystem requirement specification (SSRS) key. This is traced to Component requirement specification (CRS) key from which it is traced to Component design specification (CDS) key. CDS key is directly implemented in code. As it was mentioned earlier, right side of V-model corresponds with left side. Unit test procedure (UTPr) keys are created based on CDS keys and Component test procedure (CTPr) keys are created based on CRS keys.

Critical requirements are treated differently than non-critical requirements of embedded system. For this purpose even the code quality is evaluated specifically. The quality metrics, such as lines of code per method, static analysis criteria (McCabe, Hanson) or unit test coverage have more strictly goals, as in [10].

Basic principle of Model-based testing (MBT) (Figure 2.) is the construction of tests based on models, which describe the tested system. The used methods describe the architecture of solution and include all the requirements.
The models contain all the critical requirements about a system. This is the reason why MBT is convenient for embedded systems testing.

For this reason it is important to have a suitable representation of critical requirements of a system. This paper presents one possible solution.

This paper is divided into several sections. Chapter II describes basic principles of model-based testing of embedded systems. Chapter III presents critical knowledge representation as it is solved in existing solutions and in radiotherapy medical project.

II. MODEL-BASED TESTING OF EMBEDDED SYSTEMS

The purpose of generating tests from the test model is to produce the test repository (Figure 2.). This test repository is typically managed by a test management tool, such as HP Quality Center, IBM Rational Quality Manager or the open-source tool TestLink. The goal of such a tool is to help organize and execute test suites (groups of test cases), both for manual or automated tests.

In the MBT process, the test repository documentation is fully managed by automated generation (from the test model): documentation of the test design steps, requirements traceability links, test scripts and associated documentation are automatically provided for each test case. Therefore, the maintenance of the test repository needs to be done in the test model, as in [1].

Reference [1] uses UML for system description (Figure 3.).

This means that the developed framework implements the processing algorithm on the abstract level while cross compiler translates abstract form into physical realization for the DSP platform. The architecture of the framework was developed in object oriented UML language. The basic functionality of the framework could be expanded with the software add-ons.

Therefore the framework consists of only elementary modules which allows algorithm development and could be divided into two groups:

• Units for data processing,
• Units for data flow control.

The goal of the interactive development system in real-time was to perform validation of the development progress and functionality of the elements that were implemented on the targeted processor as well as comparing the measured results with the simulated results.

Software life cycle of embedded systems is based on multiple V-models. Testing can be divided into following phases [3]:

• Master test planning
• Testing by developers
• Testing by an independent test team

Techniques for testing embedded systems, as in [3] are:

• Risk-based test strategy
• Testability review
• Inspections
• Safety analysis
• Test design techniques
• Checklists
• Test automation

Authors in [3] present Spec Explorer modeling approach for using knowledge in testing as following:

• Redundancy scenarios derived from use cases and user stories
• Fault injection covered in the model according to a defined fault model
  o Kinds of faults (e.g. loss of connection, dead application)
  o States, in which faults might occur
  o Expected system behavior after fault
• Abstraction in test model behavior
  o Constrain model to few system features, i.e. work with partial models
  o Use trace patterns (SpecEx feature similar to regex) for scenario definition
  o Apply slicing on the model with defined test scenarios
• Abstraction in test model data
  o Define equivalence classes for values of all unbounded parameters; enumerate one value per equivalence class
  o If in doubt, one single value per parameter, i.e. one equivalence class, is often good enough

Reference [5] compares MBT and manual test creation. The main disadvantage of MBT mentioned in this article is a fact that generated tests have to be manually completed, especially in case of not fully clear requirements. This leads to a need of suitable representation of critical requirements of a system in the requirements analysis process.

III. CRITICAL KNOWLEDGE REPRESENTATION

As it was mentioned earlier, one possible approach for representing critical requirements especially from upper system levels is a set of traceable keys in whole V-model.

One of frequently used approaches for representing requirements in design phase is based on UML models. Structural as well as behavioral models can provide detailed description of architecture of developed system. Both structural and behavioral diagrams are important for implementation phase. There are many tools, which automates the prototyping, i.e. allow generation of source code parts from UML diagrams. An example of these tools can be Enterprise Architect from Sparx Systems.

As UML has the capability to model the most relevant software systems features, including critical knowledge about architecture, therefore embedded systems design has become an active area of research and several proposals have been made.

A generic framework [8] for modeling physical and logical resources of real-time systems with UML, developed by OMG, defines a unified framework to express the time, scheduling and performance aspects of a system. It is working with any runtime entity for which the services can be qualified by one or more quality of service (QoS) characteristics. They represent some aspect of the performance of a quantifiable service.

Another profile of UML for Real-Time systems is a profile [9], which extends UML notation. It defines three constructs for modeling structure: capsules, ports and connectors.

A. Critical knowledge representation in radiotherapy

This part describes how critical requirements of developed software system are handled in radiotherapy. This branch of software engineering provides software and hardware used for medical purposes. It is a complex workflow starting with CT (Computed Tomography) scans. These images serve as inputs for treatment planning system (TPS), which is used for calculation of treatment doses. Plan from TPS is sent to Oncology information system (OIS), where the exact radiation plan is created. This plan is sent to RTT (Radiation Therapist) application maintained by medical therapists. RTT communicates with the linear accelerator across the Control Console. This console acts as a mediator between RTT software and main hardware represented by linear accelerator. In this example control console is taken as embedded control. The UML-based representation of critical requirements for communication between RTT software and embedded system will be described further.

Generally, all medical software is divided into three safety classes based on IEC 62304:

- Class A – no injury or damage to health is possible
- Class B – non-serious injury is possible
- Class C – death or serious injury is possible

After code segregation on methods level all components in RTT can be assigned to one of these three classes. For us, the most important is code with class C. This code is often called as hazard code. Even code quality metrics and test cases are checked differently than for non-hazard code. If we talk about test cases, according to V-model, a critical requirement is something which is traced from system level across the requirements specification and design specification into the code. In terms of verification and validation, tracing on the left side acts as a verification. Always the lower level is verifying the upper level (e.g. code is verifying the design and design is verifying requirements). When talking about validation this is the role of the right side of V-model.

B. Experiment: Connection between RTT and Embedded System

This part represents a class from a component, which communicates with Embedded System (Control Console). This component is implemented as a state machine. Different packets are sent between RTT and embedded system based on actual state. Following states are applicable:

- Configuration upload – sent from embedded system to RTT, when connection is established. Packet data contain information about hardware configuration.
- Treatment download – treatment data is sent from RTT to embedded system. Packet data contain information from OIS.
- Treatment upload – treatment data is verified in embedded system and as a following action permission for dose delivery is granted or denied.
- Application upload – sent when treatment data is completed, i.e. specific dose was delivered by linear accelerator.

Communication is interacted by user (medical therapist), who sends the data from RTT to Control Console and manipulates with embedded system’s control.

This component is classified as class C, because in case of wrong implementation mistreatment can occur (i.e.
wrong dose is delivered to a patient) or hardware can be damaged. For this reason design must be unambiguous and detailed.

It is also a good example how UML can be used in reverse engineering. Component can be developed and maintained for years by different developers. Knowledge about architecture can be used for implementing new features such as introducing interfaces as it is very common now and was unknown in the past, when component was created.

In general, sequence diagrams in combination with state chart diagrams provide a strong mechanism for detailed architectural design. As it is in case of this example, described component is RTT is implemented as a state machine. Transitions between states are deterministic.

Unit tests in principle define modules, which should be tested. In terms of source code, these modules can be methods. Each test case, containing from several test steps, should test directly one method. It is possible to automate these cases by introducing existing frameworks as CPPUnit for C++ code or JUnit for Java code. Test cases are described in Unit Test Procedure document and those, which are related to hazard code, are traced to design specification. That’s how design specification is validated.

The purpose of integration tests is to check behavior and interoperability of several components. These tests are usually manually and are provided on the same hardware as located at customers sites. Integration test cases are described in Component Test Procedure document and again hazard test cases are traced to requirements specification for validation purposes.

C. Embedded Systems Software

In general, architecture of embedded systems can be described by a 3-tier layer model with hardware, system software and application software layers as given by Figure 5. The hardware layer represents all the major physical components located on an embedded board. The system and application software layers represent all of the software located on and being processed by the embedded system [12].

In the world of mobile devices, such as smartphones and tablets the system software is often part of device’s operating system (e.g. OS Android or iOS). It is any software that supports the applications, such as device drivers, middleware or operating system itself. Application software involves different third party applications as office or entertainment applications. An interesting domain, which is part of our research, is a medical domain. Electronic patient’s card application developed for tablets with OS Android offers several possibilities to test and verify some aspects from critical knowledge representation.

Figure 6. represents the schematic model of this application. As an Android application it has 2 main parts – graphical user interface and backend (business) logic of the application. User requests are processed by backend. If data is required, it is queried from database.

From point of users, we can identify 2 main roles – a doctor and a patient. Each of these users has different

---

**Figure 4.** Sequence diagram of communication between RTT and Embedded System

In general, sequence diagrams in combination with state chart diagrams provide a strong mechanism for detailed architectural design. As it is in case of this example, described component is RTT is implemented as a state machine. Transitions between states are deterministic.

Unit tests in principle define modules, which should be tested. In terms of source code, these modules can be methods. Each test case, containing from several test steps, should test directly one method. It is possible to automate these cases by introducing existing frameworks as CPPUnit for C++ code or JUnit for Java code. Test cases are described in Unit Test Procedure document and those, which are related to hazard code, are traced to design specification. That’s how design specification is validated.

The purpose of integration tests is to check behavior and interoperability of several components. These tests are usually manually and are provided on the same hardware as located at customers sites. Integration test cases are described in Component Test Procedure document and again hazard test cases are traced to requirements specification for validation purposes.

C. Embedded Systems Software

In general, architecture of embedded systems can be described by a 3-tier layer model with hardware, system software and application software layers as given by Figure 5. The hardware layer represents all the major physical components located on an embedded board. The system and application software layers represent all of the software located on and being processed by the embedded system [12].

In the world of mobile devices, such as smartphones and tablets the system software is often part of device’s operating system (e.g. OS Android or iOS). It is any software that supports the applications, such as device drivers, middleware or operating system itself. Application software involves different third party applications as office or entertainment applications. An interesting domain, which is part of our research, is a medical domain. Electronic patient’s card application developed for tablets with OS Android offers several possibilities to test and verify some aspects from critical knowledge representation.

Figure 6. represents the schematic model of this application. As an Android application it has 2 main parts – graphical user interface and backend (business) logic of the application. User requests are processed by backend. If data is required, it is queried from database.

From point of users, we can identify 2 main roles – a doctor and a patient. Each of these users has different
privileges. Patients have only read-only rights. The only
who can modify the plan is a doctor.

Figure 7. describes XML-based representation of
application with the list of users (roles), used environment
and list of critical requirements.

```xml
<application name="electronic patient's card">
  <roles>
    <doctor />
    <patient />
  </roles>
  <environment>
    <hardware>tablet</hardware>
    <hardware>mobile phone</hardware>
    <operating_system>Android</operating_system>
    <os_version>4.2.1</os_version>
    <database>RQLite</database>
  </environment>
  <critical_requirements>
    <requirement req_id="1">ui refresh</requirement>
    <requirement req_id="2">db response</requirement>
    <requirement req_id="3">memory management</requirement>
  </critical_requirements>
</application>
```

Figure 7. XML-based representation of application

From the point of architecture the most critical
requirements of this embedded software are related to
time response and memory management. With the
mobility user expects quick and smooth running of
application. This can be either “user-friendliness” of
graphical user interface or either quick response time
when querying some data from database.

Even if smartphones or tablets of these days have much
more available memory than older devices, memory
management of still very important aspect of creating
embedded software.

This application follows some previous research [11] in
the architecture of embedded systems, where the main
goal was put on very good real time response of the
system.

Actually, many medical software applications are
developed for embedded devices, which bring mobility to
the users. Dominant device in medical domain is iPad. For
example some OIS described in previous section used as
part of radiotherapy workflow is available for desktops as
well as for iPad tablets. Our concept brings some
innovative solutions for Android based tablets.

Implementation of this application is being performed in
Eclipse IDE with installed Android development kit using
Java programming language and SQLite database system.

Testing of this application is being performed in two
steps:

- Unit testing
- Integration testing

In unit testing phase we are focusing on modules
testing, i.e. methods of backend component and database
connectivity. It is processed using JUnit framework.
Testing of graphical user interface is part of integration
testing. Eclipse IDE provides a powerful Android
emulator (see Figure 8.) for running and testing and
Android applications. Usually when the application is
tested on real hardware (tablet in our case) it is part of
some upper level testing (e.g. system or acceptance). For
simplifying the solution, both emulator and real hardware
are part of integration testing phase.

Software requirements on developed application:

- Patient should be able to log into application
  with his user name and password
- Patient should have only read-only rights for
  checking his clinical data
- Patient can edit personal information, such as
  address, phone number or email address
- Doctor should be able to log into application
  with his user name and password
- A list of patients belonging to specific doctor
  should be enlisted for this doctor
- Doctor should be able to edit patient’s card by
  adding examinations
- Doctor should be able to add new patient to his
  list of patients or delete old one
- Application should have a functionality for
  sending data to a PC in order to print it

IV. CONCLUSION

This paper presented principles of testing of embedded
systems. It highlights the knowledge of critical
requirements of developed embedded software. According
to embedded systems these requirements can be related to
safety, reliability or user-friendliness. This knowledge is
crucial for generating test procedures. Because of different
priority and impact of these requirements test case
selection is very important. Several methods were
presented in this paper, e.g. model-based testing and also
traditional unit and integration tests from V-model. UML,
thanks to its strong extensibility, is a convenient language
for representing this set of critical requirement. The
representation of embedded systems architecture and
using this knowledge is a subject of further research.

ACKNOWLEDGMENT

This work was supported by the Slovak Research and
Development Agency under the contract No. APVV-
0008-10 and by the Scientific Grant Agency of the
Ministry of Education – VEGA Grant No. 1/0531/12 Use
of Knowledge in Software Processes.
REFERENCES


