The following full text is a preprint version which may differ from the publisher's version.

For additional information about this publication click this link.
http://hdl.handle.net/2066/34901

Please be advised that this information was generated on 2019-10-04 and may be subject to change.
Dependability for high-tech systems: an industry-as-laboratory approach

Ed Brinksma
Embedded Systems Institute, Eindhoven
University of Twente, Enschede
The Netherlands
ed.brinksma@esi.nl

Jozef Hooman
Embedded Systems Institute, Eindhoven
Radboud University Nijmegen
The Netherlands
josef.hooman@esi.nl

To appear in the Proceedings of Design, Automation & Test in Europe 2008 (DATE’08)

Abstract

The dependability of high-volume embedded systems, such as consumer electronic devices, is threatened by a combination of quickly increasing complexity, decreasing time-to-market, and strong cost constraints. This poses challenging research questions that are investigated in the Trader project, following the industry-as-lab approach. We present the main vision of this project, which is based on a model-based control paradigm, and the current status of the project results.

1. Introduction

High-tech systems by definition are constructed using cutting edge technology, and consequently embedded system technology plays a major and often even decisive role in such systems, whether they are mobile phones, HDTVs, medical equipment, cars, airplanes, etc. The integration of embedded hardware and software into larger systems has lifted the issue of dependability of the embedded components to the level of the embedding high-tech system. At that level the integral system dependability is not only affected by the dependability of its individual components, but mainly an emergent quality of the interactions between these components and the system environment. Controlling the complexity of these interactions is one of the major challenges of high-tech system design, and the presence of dependability problems in almost all application domains is a well-documented fact of life.

In this paper we want to report on a model-based approach to system dependability. Although model-based design has already been advocated for a considerable time by (mainly) the academic community as a way forward to address complex embedded systems engineering tasks, the industrial uptake is lagging behind. To ensure the practical relevance of the research, it is being carried out following an industry-as-laboratory approach [5]. This means that concrete cases are studied in their industrial context to promote the applicability and scalability of solution strategies under the relevant practical constraints.

The concrete case that we discuss is based on the collaborative research project Trader of the Embedded Systems Institute (ESI) with NXP and other academic and industrial partners on system dependability for high-volume systems. High-volume systems are characterized by the fact that because of their production in large quantities, the cost per item should be (very) low. This seriously restricts the possibility to address dependability by classical means, such as over-dimensioning of critical components. One of the main ideas in the project is to use concepts from model-based control to achieve dependability.

The rest of the paper is organized as follows: Sect. 2 contains an outline of the project, Sect. 3 outlines the model-based philosophy of the project, and Sect. 4 reports of the current status of the results. We draw our conclusions and list some future work in Sect. 5.

2. Outline of the Trader Project

In the Trader project, academic and industrial partners collaborate to optimize the dependability of high-volume products, such as consumer electronic devices. The project partners involved are: NXP Semiconductors, NXP Research, ESI, TASS, IMEC (Belgium), Twente University, the Technical University of Delft, the University of Leiden, and the Design Technology Institute (DTI) at the Eindhoven University of Technology. The project started September 2004, with a duration of five years, and includes seven PhD students and two postdocs. The so-called Carrying Industrial Partner (CIP) of this project is NXP Semiconductors,
providing the project with a focus on multimedia products. NXP has provided the problem statement and proposes relevant case studies, which in the case of Trader are taken from the TV domain. The problem statement is based on the observation that the combination of increasing complexity of consumer electronic products and decreasing time-to-market will make it extremely difficult to produce totally reliable devices that meet the dependability expectations of customers.

A current high-end TV is already a very complex device which can receive analog and digital input from many possible sources and using many different coding standards. It can be connected to various types of recording devices and includes many features such a picture-in-picture, teletext, sleep timer, child lock, TV ratings, emergency alerts, TV guide, and advanced image processing. Moreover, there is a growing demand for features that shared with other domains, such as photo browsing, MP3 playing, USB, games, databases, and networking. As a consequence, the amount of software in TVs has seen an exponential increase from 1 KB in 1980 to more than 20 MB in current high-end TVs.

Also the hardware complexity is increasing rapidly, for instance for the support of real-time decoding and processing of high-definition images for large screens and multiple tuners. To meet the hard real-time requirements a TV is designed as a system-on-chip with multiple processors, various types of memory, and dedicated hardware accelerators.

At the same time, there is a strong pressure to decrease time-to-market. To be able to realize products with many new features quickly, components developed by others have to be incorporated. This includes so-called third-party components, e.g., for audio and video standards. Moreover, there is a clear trend towards the use of downloadable components to increase product flexibility and to allow new business opportunities (selling new features, games, etc.). Given the large number of possible user settings and types of input, exhaustive testing is impossible. Also, the product must be able to tolerate certain faults in the input. Customers expect, for instance, that products can cope with deviations from coding standards or bad image quality.

Although companies invest a lot of attention and effort to avoid faults in released products, it is expected that without additional measures both internal and external faults are serious threats to product dependability. The cost of non-quality, however, is high, because it leads to many returned products, it damages brand image, and reduces market share.

The main goal of the Trader project is to improve the user-perceived dependability of high-volume products. The aim is to develop techniques that can compensate and mask faults in released products, such that they satisfy user expectations. The main challenge is to realize this without increasing development time and, given the domain of high-volume products, with minimal additional hardware costs and without degrading performance. Hence, classical fault-tolerance techniques that rely a lot on additional redundancy and resources (e.g., duplication or even triplication of hardware and software) are not suitable in this domain.

In our presentation the terminology of [1] is adopted. A failure of a system with respect to an external specification is an event that occurs when a state change leads to a run that no longer satisfies the external specification. An error is the part of the system state that may lead to a failure. For instance, a wrong memory value or a wrong message in a queue. A fault is the adjudged or hypothesized cause of an error which is not part of the system state. Examples of faults are programming mistakes (e.g., divide by zero) or unexpected input.

3. Model-Based Approach

Looking at a number of failures of consumer electronic devices, it is often the case that a user can immediately observe that something is wrong, whereas the system itself is completely unaware of the problem. Systems are often realized in a way that corresponds to the open-loop approach in control theory; for a certain input, the required actions are executed, but it is never checked whether these actions have the desired effect on the system and whether the system is still in a healthy state.

The main approach of the Trader project is to “close the loop” and to add a kind of feedback control to products. By monitoring the system and comparing system observations with a model of the desired behaviour at run-time, the system gets a form of run-time awareness which makes it possible to detect that its customer-perceived behavior is (or is likely to become) erroneous. In addition, the aim is to provide the system with a strategy to correct itself.

The main ingredients of such a run-time awareness and correction approach are depicted in Fig. 1.

![Figure 1. Adding awareness at run-time](image)

We discuss the main parts, giving examples from the TV
Chapter 4: Current Status of Trader

In this section, we give a brief description of the research activities and the current status of the Trader project. First we discuss the research on the ingredients of the global awareness vision depicted in Fig. 1: observation (Sect. 4.1), modeling system behaviour (Sect. 4.2), error detection (Sect. 4.3), diagnosis (Sect. 4.4), and recovery (Sect. 4.5). Finally, we mention research on reliability improvements during development and user perception in Sect. 4.7 and Sect. 4.6, respectively.

4.1. Observation

To observe relevant aspects of the system, both hardware and software techniques are investigated. Hardware-related work in Trader currently aims at exploiting mechanisms already available in hardware, such as the on-chip debug and trace infrastructure, to monitor values for range checking, call stacks (functions, parameters, and result values), and memory arbiters. The observation of software behaviour is mainly done by code instrumentation using aspect-oriented techniques, partly based on results from ESI-project Ideals project [6, 7]. A specialized aspect-oriented framework called AspectKoala [19] has been developed on top of the component model Koala which is used at NXP.

4.2. Modeling Desired System Behaviour

Important part of the model-based approach described in Sect. 3 is the use of a model of desired system behaviour at run-time. Experience in Trader and other ESI projects indicates that such models are usually not available in industry and that it is often difficult to obtain such models. In industrial practice, system requirements are usually distributed over many documents and databases. Hence, part of the ESI research in Trader explicitly addresses the construction of a high-level system model.

Since the TV domain is the main source of case studies in Trader, we have developed a high-level model of a TV from the viewpoint of the user. It captures the relation between user input, via the remote control, and output, via images on the screen and sound. A few first experiments indicated that the use of state machines leads to suitable models for the control behaviour of the TV. But it also revealed that it was very easy to make modeling errors, for instance, because there are many interactions between features. Examples are relations between dual screen, teletext and various types of on-screen displays that remove or suppress each other.

To allow quick feedback on the user-perceived behaviour and to increase the confidence in the fidelity of the model, Matlab/Simulink [11] is used to obtain executable models. Stateflow is exploited for the control part, whereas the streaming part of a TV is modeled by means of the Image and Video Processing toolbox of Simulink. External events can be generated by clicking on a picture of a remote control. Output is visualized by means of Matlab’s video player and a scope for the volume level. This visualization of the user view on input and output of the model turned out to be very useful to detect modeling errors and undesired feature interactions. In addition, we investigate the possibilities of
formal model-checking and test scripts to improve model quality.

4.3. Error Detection

Various techniques for error detection are investigated such as hardware-based deadlock detection and range checking. An approach which checks the consistency of internal modes of components turned out to be successful to detect teletext problems due to a loss of synchronization between components [17].

To enable quick experimentation with model-based error detection, we have developed a framework which allows the use of models at run-time. The framework has been implemented on top of Linux, to comply with the trend towards open-source software and the use of Linux in TVs. In the framework, one can include a particular System Under Observation (SUO) and a specification model of the desired system behaviour. The design of the awareness framework is shown in Fig. 2. The SUO and the awareness monitor are included easily, allowing quick experiments with different possibilities of Stateflow. The generated C-code can be included by using the code generation possibilities of Stateflow. The generated C-code can be included easily, allowing quick experiments with different models. It is executed using the Model Executor component, based on event notifications from the Input Observer. Information about relevant input and output events is stored in the Configuration component. The Comparator component compares relevant model output with system output which is obtained from the Output Observer. The Controller initiates and controls all components, except for the Configuration component which is controlled by the Model Executor.

Experiments with earlier versions of the framework indicated that the Comparator should not be too eager to report errors; small delays in system-internal communication might easily lead to differences during a short time interval. Hence, in the current framework the user of the framework can specify, for each observable value: (1) a threshold for the allowed maximal deviation between specification model and system, and (2) a maximum for the number of consecutive deviations that are allowed before an error will be reported.

Another relevant parameter is the frequency with which time-based comparison takes place. This can be combined with event-based comparison by specifying in the specification model when comparison should take place and when not (e.g., when the system is in an unstable state between certain modes). Observe that we have to make a trade-off between taking more time to avoid false errors and reporting errors fast to allow quick repair.

Related to our approach is a method to wrap COTS components and monitor them using specifications expressed as a UML state diagrams is presented in [16]. Other related work consists of assertion-based approaches such as runtime verification [4]. For instance, monitor-oriented programming [3] supports run-time monitoring by integrating specifications in the program via logical annotations. In our approach, we aim at minimal adaptation of the software of the system, to be able to deal with third-party software and legacy code. Moreover, we also monitor real-time properties, which are not addressed by the techniques cited above. Closely related in this respect is the MaC-RT system [15] which also detects timeliness violations. Main difference with our approach is the use of a timed version of Linear Temporal Logic to express requirement specifications, whereas we use executable timed state machines to promote industrial acceptance and validation.

4.4. Diagnosis

The diagnoses techniques developed within Trader are based on so-called program spectra [20]. The approach has already been applied to a few examples in the TV domain. As an illustration, we describe one of the first experiments on TV software in which a teletext error has been injected. First the C code is instrumented to record which blocks are executed. In the example there were 60 000 blocks. Next, for each sequence of key presses, a so-called scenario, for each block it is recorded whether it has been executed or not between two key presses. This leads to a vector, a so-called spectrum, for each block. In our example it turns out
that during a scenario of 27 key presses 13 796 blocks were executed. Moreover, based on some error detection mecha-
nism, it is recorded for each key press whether it leads to error or not. In the example, this leads to an error vec-
tors of length 27. Next, the similarity between the error vector and the spectra is computed. Finally, the blocks are
ranked according their similarity. In the particular experiment with the teletext error, the block which contains the
fault appeared on the first place in the ranking. Also in other case studies the application results of this technique are en-
couraging.

4.5. Recovery

Part of the recovery research concentrates on load bal-
ancing. Project partner IMEC has demonstrated the possi-
bility to migrate an image processing task from one proces-
sor to another, which leads to improved image quality in
case of overload situations (e.g., due to intensive error cor-
rection on a bad input signal). NXP Research investigates
the possibility to make memory arbitration more flexible
such that it can be adapted at run-time to deal with problems
concerning memory access. At Twente University, a frame-
work for partial recovery has been developed which allows
independent recovery of parts of the system, the so-called
recoverable units. The framework includes a communica-
tion manager, which controls the communication between
recoverable units, and a recovery manager, which executes
the recovery actions such as killing and restarting units. To
realize these concepts, a reusable fault tolerance library has
been implemented. A few first experiments in the multi-
media domain show that after some refactoring of the sys-
tem, independent recovery of parts of the system is possible
without large overhead.

4.6. User Perception

The user perception of reliability is addressed by project
partner DTI. The aim is to capture user-perceived failure se-
verity, to get an indication of the level of user-irritation
caused by a product failure. By means of controlled ex-
periments with TV users, the impact of characteristics such
as product usage, user group, and function importance is
investigated. During experiments, it turned out that also
failure attribution has a significant impact. For instance,
users, when asked, rank both image quality and a motor-
ized swivel, which can be used to turn the TV, as important.
Under observation, however, users often turn out to be very
tolerant concerning bad image quality (which is attributed
to external sources), but get irritated if the swivel doesnot
work correctly.

4.7. Improvements During Development

Part of the Trader research is also related to dependabil-
ity improvements during development. This includes the
use of code analysis to prioritize the warnings of a software
inspection tool such as QA-C [2] and reliability analysis at
the architectural level [18]. The stress testing approach of
TASS artificially takes away shared resources, such as CPU
or bus bandwidth, to simulate the occurrence of errors or the
addition of an additional resource user. The study of the ef-
effect of such overload situations on the system behaviour and
its fault-tolerant mechanisms has shown to be very useful in
the TV domain. A so-called CPU eater, which consumes
CPU cycles at the application level in software, is already
included in the current development software and can be
activated by system testers.

5. Conclusion and future Work

Although the Trader project has still some time to go, it
is already clear that its particular model-based approach to
system dependability is very promising. The use of models
as system components to give the system a certain capacity
to monitor and correct its behaviour, implements ideas from
feedback control at the level of integrated systems. It con-
stitutes a paradigm switch from the best-effort, open-loop
approach that is traditional in software-related design, to
the closed-loop control-based approach. The latter is much
more suitable for the reality of high-tech systems in which
errors are unavoidable emergent features of the system com-
plexity.

The concept of model-based system level control is also
quite flexible, in the sense that one can vary between light-
weight models with limited corrective capacities, and more
elaborate models with stronger feedback mechanisms. In
the high-volume context, the constraint to minimize over-
head is limiting factor. Certainly, much more research will
be needed to obtain a more complete understanding of the
potential and limitations of this approach in the a priori vast
range of different application domains.

The choice for an industry-as-laboratory format for the
Trader project has helped a lot in focussing on techniques
and approaches that have a high potential for being ab-
sorbed by industry. Already now, some of the intermedi-
ate results have found their way into industry. We firmly
believe in the potential of this research format to achieve a
productive combination between real research and innova-
tion.

Future activities in the Trader project will address further
development of the awareness framework. Our Linux-based
awareness framework, has been validated by means of
model-to-model experiments. That is, we have compared a
specification model with code generated from models of the
SUO. Currently, the framework is used for awareness experiments with the open source media player MPlayer [12], investigating both correctness and performance issues. Next our approach will be applied in the TV domain at NXP, following the industry-as-lab paradigm. Important topic of research concerns the optimal integration of various techniques for observation, error detection, diagnosis, and recovery.

In parallel, the model-based run-time awareness concept is also exploited in the domain of printer/copiers at the company Océ in the context of the ESI-project Octopus [8], which started recently.

References