MŰEGYETEM 1782

Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Measurement and Information Systems

Hierarchical runtime verification for critical cyber-physical systems

Scientific Students' Associations Report

Authors:

László Balogh
Flórián Dée
Bálint Hegyi

Supervisors:

dr. István Ráth
dr. Dániel Varró
András Vörös

2015.

Contents

Összefoglalás Ipari becslések szerint 2020-ra 50 milliárdra nő a különféle okoseszközök száma, amelyek egymással és velünk kommunikálva komplex rendszert alkotnak a világhálón. A szinte korlátlan kapacitású számítási felhőbe azonban az egyszerű szenzorok és mobiltelefonok mellett azok a kritikus beágyazott rendszerek – autók, repülőgépek, gyógyászati berendezések - is bekapcsolódnak, amelyek működésén emberéletek múlnak. A kiberfizikai rendszerek radikálisan új lehetőségeket teremtenek: az egymással kommunikáló autók baleseteket előzhetnek meg, az intelligens épületek energiafogyasztása csökken.

A hagyományos kritikus beágyazott rendszerekben gyakorta alkalmazott módszer a futási idejű ellenőrzés. Ennek célja olyan ellenőrző programok szintézise, melyek segítségével felderíthető egy kritikus komponens hibás, a követelményektől eltérő viselkedése a rendszer működése közben.

Kiberfizikai rendszerekben a rendelkezésre álló számítási felhő adatfeldolgozó kapacitása, illetve a különféle szenzorok és beavatkozók lehetővé teszik, hogy több, egymásra hierarchikusan épülő, különböző megbízhatóságú és felelősségű ellenőrzési kört is megvalósíthassunk. Ennek értelmében a hagyományos, kritikus komponensek nagy megbízhatóságú monitorai lokális felelősségi körben működhetnek. Mindezek fölé (független és globális szenzoradatokra építve) olyan rendszerszintű monitorok is megalkothatók, amelyek ugyan kevésbé megbízhatóak, de a rendszerszintű hibát prediktíven, korábbi fázisban detektálhatják.

A TDK dolgozatban egy ilyen hierarchikus futási idejű ellenőrzést támogató, matematikailag precíz keretrendszert dolgoztunk ki, amely támogatja (1) a kritikus komponensek monitorainak automatikus szintézisét egy magasszintű állapotgép alapú formalizmusból kiindulva, (2) valamint rendszerszintű hierarchikus monitorok létrehozását komplex eseményfeldolgozás segítségével. A dolgozat eredményeit modellvasutak monitorozásának (valós terepasztalon is megvalósított) esettanulmányán keresztül demonstráljuk, amely többszintű ellenőrzés segítségével képes elkerülni a vonatok összeütközését.

Abstract According to industrial estimates, the number of various smart devices - communicating with either us or each other - will raise to 50 billion, forming one of the most complex systems on the world wide web. This network of nearly unlimited computing power will not only consist of simple sensors and mobile phones, but also cars, airplanes, and medical devices on which lives depend upon. Cyber-physical systems open up radically new opportunities: accidents can be avoided by cars communicating with each other, and the energy consumption of smart buildings can be drastically lower, just to name a few.

The traditional critical embedded systems often use runtime verification with the goal of synthesizing monitoring programs to discover faulty components, whose behaviour differ from that of the specification.

The computing and data-processing capabilities of cyber-physical systems, coupled with their sensors and actuators make it possible to create a hierarchical, layered structure of high-reliability monitoring components with various responsibilities. The traditional critical components' high-reliability monitors' responsibilities can be limited to a local scope. This allows the creation of system-level monitors based on independent and global sensory data. These monitors are less reliable, but can predict errors in earlier stages.

This paper describes a hierarchical, mathematically precise, runtime verification framework which supports (1) the critical components' monitors automatic synthetisation from a high-level statechart formalism, (2) as well as the creation of hierarchical, system-level monitors based on complex event-processing. The results are presented as a case study of the monitoring system of a model railway track, where collisions are avoided by using multi-level runtime verification.

Chapter 1

Introduction

Chapter

Chapter 2

Background

Chapter

Chapter 3

Overview

Chapter

Chapter 4

Runtime verification of embedded systems

4.1 Intro

A statechart language was created to enable the high level design, verification, and monitoring of complex systems. The aim was to use a simple and straightforward syntax to keep the language's learning curve gentle. Statecharts were chosen as they are used widely for modeling in various branches of engineering.

4.2 Goal

4.2.1 Simple, generic, useable

4.2.2 Verification

4.2.3 Monitor generation

4.3 Why not upgrade previous solutions

Validation software is... Many software is available for code generation. Unfortunately the available solutions either provide poor quality code or have a limited syntax, thus making the creation and understanding of the models more time consuming than necessairy. Our approach was to generate easily readable, extendable, object oriented code that can run in a limited resource environment.

4.3.1 Parametric statechart declaration

The language allows a specification to consist of multiple statecharts. This feature led to of the main strengths of the language: the definition of statechart templates, which can be parametrically instantiated multiple times. This results in short descriptions for otherwise complex, homogenious systems. Statechart parameters can be of any type supported by the TTMC::Constraint language. Separate statecharts can communicate with each other using signals or global variables.

4.3.2 Parametric signals

Signals can also be parameterized with any integer type variable. These parameters then can be used to discriminate between signals with the same name, which also results in more readable code, allowing transitions to use the same signal as their trigger.

4.4 Statechart specification language for enginneering models

4.4.1 The specification

A specification can consist of multiple statecharts. Statechart definitions must be in the form of
statechart NAME(...) ...
, where the
...
part contains the description of the statechart. The braces are optional and are only needed for the parameters of statechart templates. For statechart declarations, the description can be omitted. Each specification must have at least one defined statechart. Parameterized statecharts can be created from existing templates by providing a value for each parameter.

4.4.2 Variables

Variables can either be global (accessible to all statecharts) or local (bound to a single statechart in which they were declared). Many types are supported chapter characters, integers, doubles, etc... For a complete list, see
appendix4TTMCConstraint
. The variable declaration is in the form of
global|local var NAME : TYPE
, where global or local denotes the scope of the variable.

4.4.3 Expressions and assignments

Variables can be used in expressions. Expressions can have an arbitrarily complex structure within the limits of the
TTMC::Constraint
language. This allows for, among others, the use of array indexing, parenthesis, and common operators in programming languages (+, -, *, /, modulo, ...). Assignments left hand sides are a single variable while their right hand side is an expression. Logical expressions using operators are also available (for example expressions using comparison operators). Each expression is a mixature of variables, constants, and operators. For a full reference, see
TTMC::Constraint
.

4.4.4 Parametric signals

Statecharts can communicate with the outside world and each other using signals. As such, these signals are declared directly in the specification and not in the statecharts themselves. Signals can be used with a single integer parameter (which can be either a constant or a variable). This allows for much simpler syntax when dealing with communication, as a statechart can raise a signal and pass a value simultaneusly. It also leaves room for a later expansion to a token based automata with reentry.

4.4.5 Timeouts

Raising a signal can be offset by a certain amout of time. For the formal model, the value is measured in units, for the monitors, this value corresponds to the milliseconds elapsed since the timeout was set. Timeouts and signals can be used interchangeably.

4.4.6 Entry and exit actions

4.4.7 Regions

Statecharts are structured by the usage of regions. Regions have both states and transitions, and play a fundament part in the scoping of elements. The syntax for regions is:
region NAME ...
Each region must contain at least an initial state for the model to be valid.

4.4.8 Transitions

Transition actions

Transition guards

Transition timing

4.4.9 State nodes

Each region can contain multiple state nodes. A state node can either be a state or a pseudo state. States create the base structure of the model, while pseudo states help to describe the functionality. Pseudo states can either be an initial, a fork, a join, or a choice state.

States

States can either be atomic states or composite states. An atomic state is a state which does not contain inner regions. All states can contain entry and exit actions. Composite states contain one or more inner regions, each with at least an inner initial state. A state's parent is it's containing region's containing state, or if that region does not have containing state, the region's containing statechart. The inverse of this relation is called of it's child. Composite states help maintaining a clean model by the introduction of hierarchy, allowing common actions to be described in a parent state.

Fork and join states

A fork state is a pseudo state that has a single

Composite state

4.4.10 Error states, propagation

4.5 Formal representation

4.5.1 Signals

4.5.2 Variables

4.5.3 Expressions

4.5.4 States

4.5.5 Timouts

4.5.6 Transitions

4.6 Accepting monitor

4.6.1 Variables

4.6.2 Signals

4.6.3 Timeouts

4.6.4 States

4.6.5 Transitions

4.7 Implementation

4.7.1 Other utility classes

4.7.2 Timing (clock of the monitor)

4.7.3 Interface, signal pushing

4.8 Summary

Chapter 5

Complex event processing

Chapter

5.1 Formal Intro of the Event Automatons

5.1.1 Current Formalisms

Quantified Event Automaton

Calendar Event Automaton

5.1.2 Our Formalism

Ezeknek nyilvan fancybb nev kell, csak errol fog szolni.

Things which are the same as in one of the previous formalisms

Things which are not the same (changed?)

Nondeterminism -> Deterministic behaviour

Timing

5.2 Examples of Event Processing

5.2.1 File System

5.2.2 Mars Rover Tasking

5.3 Implementation

5.3.1 Metamodel

5.3.2 Executor

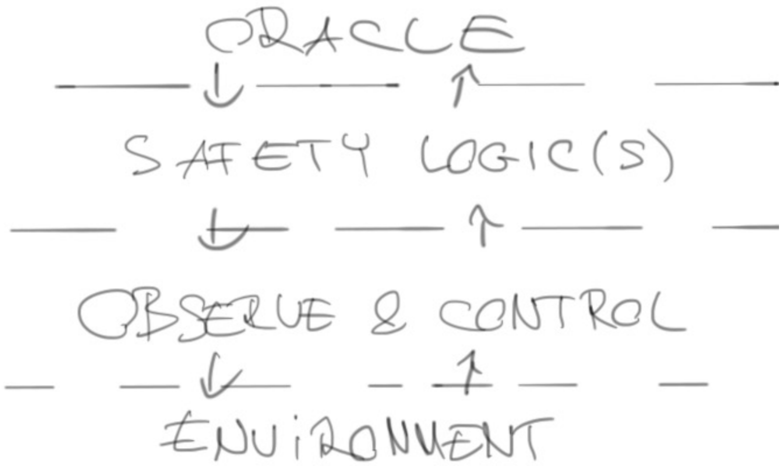
Chapter 6

Case study

6.1 Overview

The goal of this chapter is to present a case study of a hierarchical runtime verification technology based on the previous chapters. The motive of this study is the related report from 2014 [1], which goal was a distributed, model based security logic. Their case study called the *Model Railway Project*, and our study integrates their work, and finds ways to integrate it with other sensor sources for a hierarchical, more reliable security logic.

Because of this close relation to this railroad project, our focus will stay on railroad technologies and standards, it's important to notice our solution is a general approach for any critical system. This approach is based on controllability, observability, and hierarchy to increase a system general reliability, even when some of it's components are failing.



6.2 Architecture

6.2.1 Total view

Itt átvesszük a hardware alapjait, hogy egyáltalán miről van szó, hogy néz ki, mit tud.

6.2.2 Hardware

Itt az arduino alapú vezérlést emelném ki röviden, az ezzel felmerült problémákat, illetve a jövőbeli fejlesztések rövid bemutatását.

6.3 Concept

A koncepció magyarázata, szép összefoglaló ábrával, és annak az összefoglalása, hogy mit tudunk ezzel elérni.

6.4 Computer vision as a source of information

6.4.1 Hardware

In case of a computer vision (CV) based approach, it is critical to choose the appropriate hardware. We had two parameters in the selection of the camera: height above the board, and FOV.

> **Definition 6.1** Field of View (FOV) is the extent of the observable world that is seen at any given moment. See Figure 6.1 for visual explanation.

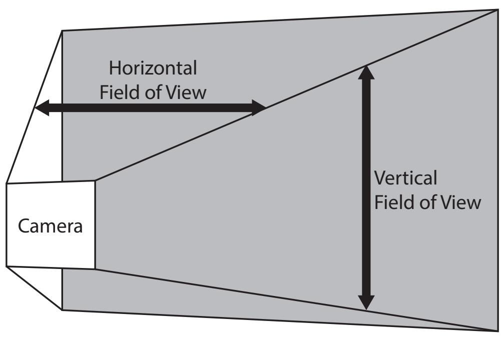


Figure 6.1 Visual explanation of FOV

These parameters are coupled, the higher the camera the less FOV we need. Most of the cameras on the market have horizontal FOV values approximately 60°.

Example 6.1 The board is 2.8 m wide. So if we assume we have a camera with a 60° FOV, using the result of eq. (6.1), we need to place the camera 242 centimeters high. This would be is impossible to realize with average ceiling heights.

$$242.48 \text{ cm} = \frac{140 \text{ cm}}{\tan(30°)} \tag{6.1}$$

Our chosen FOV became 120°, because the cameras with these FOV values are inexpensive, and easy to find. With a placement height of 120 cm, we can fully assemble the project virtually in any room.

6.4.2 Introducing to OpenCV

One key point of this study from technological viewpoint is computer vision. It is a non intrusive add-on to the existing hardware, which allows us to monitor the board with fairly big precision and reliability, if the correct techniques and materials are used.

We needed a fast, reliable, efficient library to use with the camera, and develop the detection algorithm. Our choose was the OpenCV[1] library, which is an industry leading, open source computer vision library. It implements various algorithms with effective implementation in mind e.g., using the latest streaming vector instruction sets. The main programming language – and what we used – is C++, but it has many binding to other popular languages like Java, and Python.

6.4.3 Marker design

One of the steps of the CV implementation was the design of the markers, which should provide an easy detection, and identification of the marked objects.

The first step was to consider the usage of an external library, named ArUco[2]. This library provides the generation (see Figure 6.2), and detection library of markers.

The problem with the library was the lack of tolerance in quality, and motion blur. Other limiting factor is the shape of the marker. A square marker scaled up to provide good visibility for the camera 120 cm above the board extends greatly over the width of a railway car. Because these negative properties of the existing libraries, we implemented a marker detection algorithm for our needs.

[1] `http://opencv.org/`

[2] `http://www.uco.es/investiga/grupos/ava/node/26`

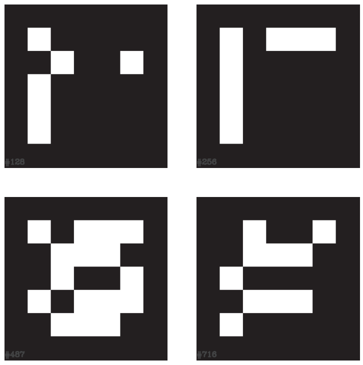


Figure 6.2 Example markers generated by ArUco

After the implementation was in our hands, we could make markers which suits our needs. The optimal marker covers the railroad car. This means the marker is narrow and long to fill the top dimensions of the car, but not exceed it.

As explained in Section 6.4.4, circular patterns are well suited for these applications. The final design consists two detection circle, and a color circle for identification between the detection circles.

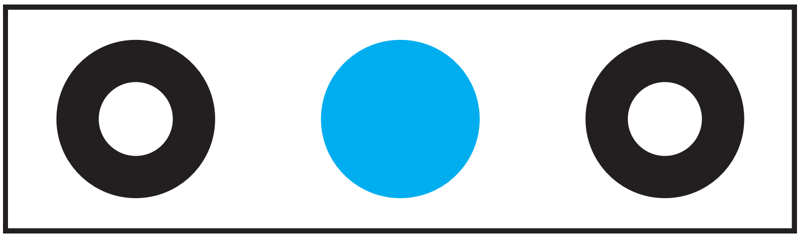


Figure 6.3 The final marker design with a blue ID circle

6.4.4 Mathematical solution for marker detection

Dimensions mentioned in Section 6.2.1 cause various environmental lighting conditions across the board. This is a challenging problem, because we must detect every circle despite we defined it as a perfect black circle, and every maker have its own lighting condition.

This problem, and the fact that these markers have perspective distortion when they are near to the visible region of the camera motived us to develop a processing technique coming from controlling theory.

This method is the commonly used technique of transforming and processing a signal – in our case a picture – in frequency domain.

Convolution method

Our method is based on the convolution of two bitmap images, one from the camera, and one generated pattern.

Definition 6.2 The convolution of image functions I_1, I_2 is:

$$I_1 * I_2 = \mathscr{F}^{-1}(\mathscr{F}(I_1) \cdot \mathscr{F}(I_2)) \tag{6.2}$$

As eq. (6.2) denoted, we can multiply two spectrums element-wise, and apply an inverse Fourier transform to get the convoluted image. If one image is the pattern, the other image is the raw[3], applying the convolution results in an image where every pixel represents a value how much the two spectra matches.

Pattern bitmap properties

While generating the pattern bitmap, we should consider it's properties:

- The bitmap must be the same size as the raw image we applying it to.
- While the pattern can be placed anywhere in the bitmap, there will be an offset from the origin to our pattern center point. For example if we put the circular pattern in the upper left corner (let the upper left corner be the origin of the bitmap), the matching pattern will have an offset error of $[r, r]$ (Figure 6.5). Instead of correcting this offset, we can use the periodicity of the Fourier transform, therefore we can put our pattern in the corners. In case of this placement, the center will be the out patterns imaginary[4] center like the marker in Figure 6.6.

The pattern itself needs to be generated with values according to the shape we would like to match (Figure 6.6). The raw pixels are multiplied by this value. The meaning of these values in the bitmap are the following:

- $\boldsymbol{value = 0}$: Doesn't affect the match.
- $\boldsymbol{value > 0}$: The multiplied raw pixel summed positively to the result of the convolution.
- $\boldsymbol{value < 0}$: The multiplied raw pixel summed negatively to the result of the convolution.

[3]In our application raw (or raw image) means the unprocessed image from the camera
[4]Naturally our shapes could be more complicated, where we have our imaginary center of the shape

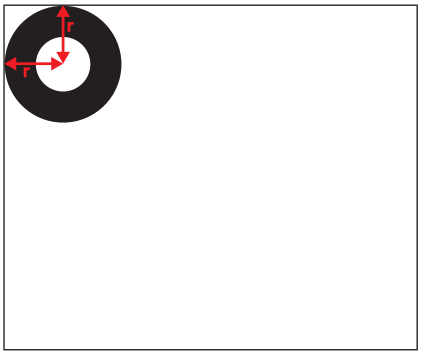


Figure 6.4 A test pattern

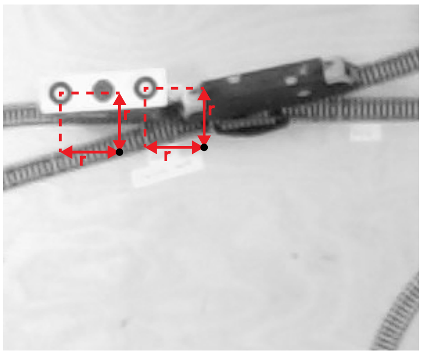


Figure 6.5 A example of offset error

Figure 6.6 Pattern bitmap placement and value example

6.4.5 Software

CPU implementation

Az első implementáció gyors bemutatása.

CUDA implementation

A GPU által gyorsított verzió bemutatása, mit tapasztaltunk ennek során, hogyan segített a fejlesztésben.

6.5 Physical - logical mapping

6.5.1 Elements of physical mapping

Az általunk használható architektúra részletes bemutatása.

6.5.2 Elements of logical mapping

Azoknak az elemeknek, elképzeléseknek az áttekintése, amik szerepelhetnek a modellünkben, még teljesen független módon.

6.5.3 Introducing to EMF

Az EMF bemutatása röviden.

6.5.4 Building the EMF model

Az elkészült EMF metamodell bemutatása, és összevetése az elképzelésekkel.

6.5.5 Introducing the IncQuery

Az IncQuery bemutatása.

6.5.6 Building the IncQuery patterns

A biztonsági lokikai patternek bemutatása.

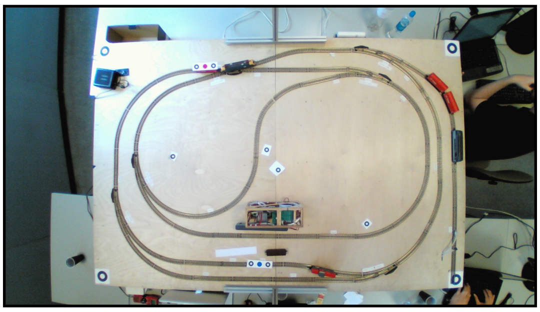
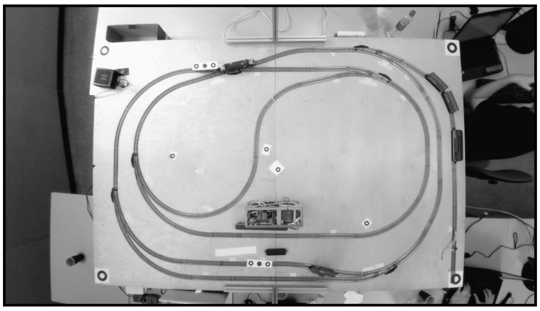
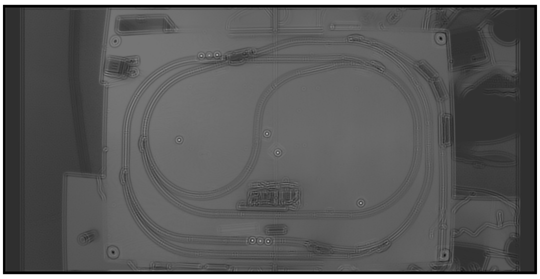
Stage #	Description	
Stage 1	Loading an image from the camera	
Stage 2	Convert the image to grayscale	
Stage 3	Convolve the image with the pattern	
Stage 4	Apply a thresold	

Table 6.1 Computer vision processing pipeline

Chapter 7

Conclusion

Chapter

References

[1] Mázló Zsolt Horváth Benedek Konnerth Raimund-Andreas. *Elosztott biztonságkritikus rendszerek modellvezérelt fejlesztése*. Tech. rep. Budapest University of Technology et al., 2014.