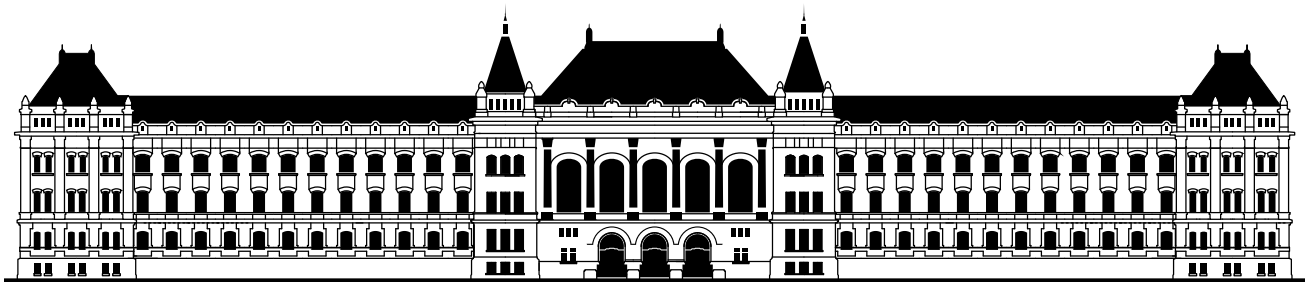
IEEE

INSTRUMENTATION AND MEASUREMENT & ENGINEERING
IN MEDICINE AND BIOLOGY JOINT CHAPTER

PROCEEDINGS
OF THE
23RD PHD MINI-SYMPOSIUM

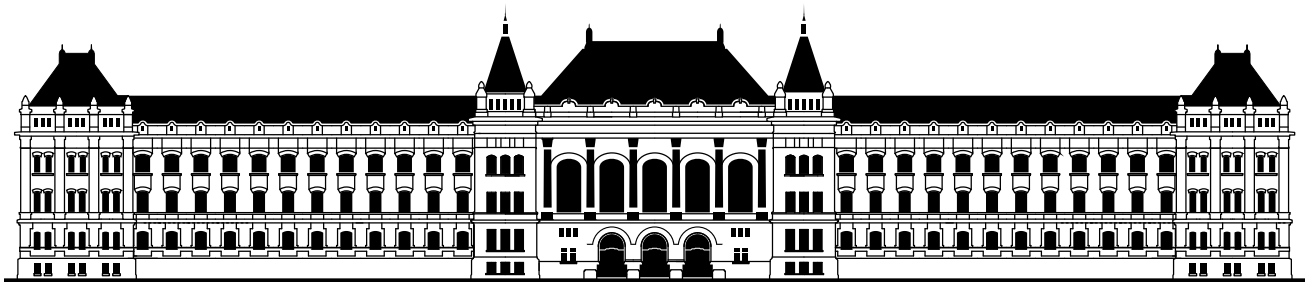
FEBRUARY 8–9, 2016



BUDAPEST UNIVERSITY OF TECHNOLOGY AND ECONOMICS
DEPARTMENT OF MEASUREMENT AND INFORMATION SYSTEMS

PROCEEDINGS
OF THE
23RD PHD MINI-SYMPOSIUM

FEBRUARY 8–9, 2016
BUDAPEST UNIVERSITY OF TECHNOLOGY AND ECONOMICS
BUILDING I



BUDAPEST UNIVERSITY OF TECHNOLOGY AND ECONOMICS
DEPARTMENT OF MEASUREMENT AND INFORMATION SYSTEMS

© 2016 by the Department of Measurement and Information Systems
Head of the Department: Dr. Tamás Dabóczi

Conference chairman:
Béla Pataki

Organizers:
Ágnes Salánki
Oszkár Semeráth
Gábor Szárnyas
Tamás Virosztek

Homepage of the Conference:
`http://minisy.mit.bme.hu/`

Sponsored by:

IEEE Instrumentation and Measurement & Engineering
in Medicine and Biology Joint Chapter

Schnell László Foundation

ISBN 978-963-313-220-3

FOREWORD

This proceedings is a collection of the lectures of the 23rd Minisymposium held at the Department of Measurement and Information Systems of the Budapest University of Technology and Economics. In the previous years the main purpose of these symposia was to give an opportunity to the PhD students of our department to present a summary of their work done in the preceding year. It is an interesting additional benefit, that the students get some experience: how to organize such events. Beyond this actual goal, it turned out that the proceedings of our symposia give an interesting overview of the research and PhD education carried out in our department. This year the scope of the Minisymposium has been widened; foreign partners and some best MSc students are also involved.

The lectures reflect partly the scientific fields and work of the students, but we think that an insight into the research and development activity of our and partner departments is also given by these contributions. Traditionally our activity was focused on measurement and instrumentation. The area has slowly changed, widened during the last few years. New areas mainly connected to embedded information systems, new aspects e.g. dependability and security are now in our scope of interest as well. Both theoretical and practical aspects are dealt with.

The lectures are at different levels: some of them present the very first results of a research, others contain more new results. Some of the first year PhD students have been working on their fields only for half a year. There are two types of papers. One is a short independent publication; it is published in the proceedings. The other is simply a summary of the PhD student's work. This second one is intended to give an overview of the research done during the last year; therefore, it could contain shorter or longer parts of the PhD student's other publications. It does not necessarily contain new results, which have not been published earlier. It is clearly indicated in each paper that which category it belongs to. To avoid copyright conflicts, these papers are not published in the proceedings. Anyone interested, please contact the author.

During this twenty-two-year period there have been shorter or longer cooperation between our department and some universities, research institutes, organizations and firms. Some PhD research works gained a lot from these connections. In the last year the cooperation was especially fruitful with the European Organization for Nuclear Research (CERN), Geneva, Switzerland; Vrije Universiteit Brussel Dienst ELEC, Brussels, Belgium; Robert Bosch GmbH., Stuttgart, Germany; Department of Engineering, Università degli Studi di Perugia, Italy; Department of Medical Engineering and Physics, Riga Technical University, Latvia; National Instruments Hungary Kft., Budapest; IEEE Instrumentation and Measurement Society & Engineering in Medicine and Biology Society Joint Chapter, IEEE Hungary Section.

We hope that similarly to the previous years, also this Minisymposium will be useful for the lecturers, for the audience and for all who read the proceedings.

Budapest, January, 2016

Béla Pataki
Chairman of the PhD
Mini-Symposium

LIST OF PARTICIPANTS

Participant	Supervisor	Programme
Birpoutsoukis, Georgios	Schoukens, Johan	PhD
Darvas, Dániel	Majzik, István	PhD
Debreceni, Csaba	Varró, Dániel	PhD
Farkas, Rebeka	Vörös, András	MSc
Hadházi, Dániel	Horváth, Gábor	PhD
Hajdu, Csaba	Dabóczi, Tamás	PhD
Honfi, Dávid	Micskei, Zoltán	PhD
Maginecz, János	Szárnyas, Gábor	MSc
Molnár, Vince	Majzik, István	PhD
Nagy, András Szabolcs	Varró, Dániel	PhD
Pasku, Valter	Carbone, Paolo	PhD
Pataki, András	Horváth, Gábor	PhD
Piksis, Martins	Dekhtyar, Yuri	PhD
Renczes, Balázs	Kollár, István	PhD
Salánki, Ágnes	Pataricza, András	PhD
Semeráth, Oszkár	Varró, Dániel	PhD
Szárnyas, Gábor	Varró, Dániel	PhD
Sólyom, Anna Alexandra	Nagy, András Szabolcs	MSc
Tóth, Tamás	Majzik, István	PhD
Virosztek, Tamás	Kollár, István	PhD
Várszegi, Kristóf	Pataki, Béla	MSc

PAPERS OF THE MINI-SYMPOSIUM

Author	Title	Page
Birpoutsoukis, Georgios	Nonparametric Volterra Kernel Estimation Using Regularization	*
Darvas, Dániel	Generic Representation of PLC Programming Languages for Formal Verification	6
Debreceni, Csaba	Automated Model Merge by Design Space Exploration	*
Farkas, Rebeka	Towards Efficient CEGAR-Based Reachability Analysis of Timed Automata	10
Hadházi, Dániel	Partial Volume Artifact in MITS Reconstructed Digital Chest Tomosynthesis	14
Hajdu, Csaba	On-Line Detector Connectivity Check Based on a Parasitic Signal Coupling	18
Honfi, Dávid	Generating Unit Isolation Environment Using Symbolic Execution	22
Maginecz, János	Sharded Joins for Scalable Incremental Graph Queries	26
Molnár, Vince	Evaluation of Fault Tolerance Mechanisms with Model Checking	30
Nagy, András Szabolcs	State Coding Techniques in Rule-Based Design Space Exploration	*
Pasku, Valter	A Low Cost Magnetic-Field Based Indoor Positioning System	34
Piksis, Martins	Novel Technique for Radiation Dose Visualization in Large Space	38
Pataki, András	Exploratory Data Analysis of Printed Circuit Boards	*
Renczes, Balázs	A Novel Evaluation Technique for Least Squares Sine Wave Fitting Algorithms	*
Salánki, Ágnes	Data Analysis Based Qualitative Modeling	*
Semeráth, Oszkár	Iterative and Incremental Model Generation by Logic Solvers	
Szárnyas, Gábor	Towards a Macrobenchmark Framework for Performance Analysis of Java Applications	*
Sólyom, Anna Alexandra	Swarm Intelligence Meets Rule-Based Design Space Exploration	42
Tóth, Tamás	Formal Modeling of Real-Time Systems with Data Processing	46
Virosztek, Tamás	Examination of Limits of Parameter Estimation Based on Quantized Data	*
Várszegi, Kristóf	Overview and Prospects of Brain-Computer Interface Technology for Prosthetic Limbs	50

The research reports indicated by * are not published in this volume. To gain access, please contact the organizers or the authors.

Generic Representation of PLC Programming Languages for Formal Verification

Dániel Darvas*†, István Majzik* and Enrique Blanco Viñuela†
*Budapest University of Technology and Economics, Department of Measurement and Information Systems
Budapest, Hungary, Email: {darvas,majzik}@mit.bme.hu
†European Organization for Nuclear Research (CERN), Beams Department
Geneva, Switzerland, Email: {ddarvas,eblanco}@cern.ch

***Abstract*—Programmable Logic Controllers are typically programmed in one of the five languages defined in the IEC 61131 standard. While the ability to choose the appropriate language for each program unit may be an advantage for the developers, it poses a serious challenge to verification methods. In this paper we analyse and compare these languages to show that the ST programming language can efficiently and conveniently represent all PLC languages for formal verification purposes.**

I. Introduction and Background

Programmable Logic Controllers (PLCs) are widely used for various control tasks in the industry. As they often perform critical tasks – sometimes PLCs are even used in safety-critical settings up to SIL3 – the verification of these hardware-software systems is a must. Besides the common testing and simulation methods, formal verification techniques, such as model checking are increasingly often used.

The corresponding IEC 61131 standard defines five PLC-specific programming languages: Instruction List (IL), Structured Text (ST), Ladder Diagram (LD), Function Block Diagram (FBD) and Sequential Function Chart (SFC) [1]. It is out of the scope to discuss the features of these languages in detail, but a simple example in Figure 1 shows the different flavours of these languages. The first four example program excerpts are *execution equivalent*, i.e. for all possible starting (input and retained) variable valuations, the results of these programs are the same variable valuations. The SFC example is different from the others, as this is a special-purpose language for structuring complex applications.

This variety of languages responds to the fact that PLCs are used in different settings and programmed by people with various backgrounds. This is an advantage for the developers, but an important challenge for the verification. The languages can be freely mixed, e.g. a function written in IL can call an ST function. To provide a generally applicable formal verification solution, all these languages should be supported.

A. Motivation

Our practical motivation lies in the PLCverif formal verification tool and its workflow [2], [3]. The PLCverif tool provides a way for PLC program developers to apply model checking to their implementation. This allows to check the satisfaction of various state reachability, safety and liveness requirements. The inputs of the model checking workflow are the source code and the requirements formalized using verification patterns. At the moment, programs (or program units) written in the Siemens variant of ST or SFC are supported. These inputs are convenient for the users not familiar with formal verification methods. PLCverif automatically generates temporal logic expressions from the pattern-based requirements, parses the input code, builds a simple, automata-based intermediate verification model, calls the chosen external model checker tool (e.g. nuXmv), and presents the results in a simple, self-contained format to the user. The tool is in use at the European Organization for Nuclear Research (CERN) to check critical control programs [4]. While most of the PLC programs are written in ST at CERN, in special cases (e.g. safety-critical applications) restrictions forbid the use of ST. To make PLCverif generally applicable, all five PLC programming languages should be supported.

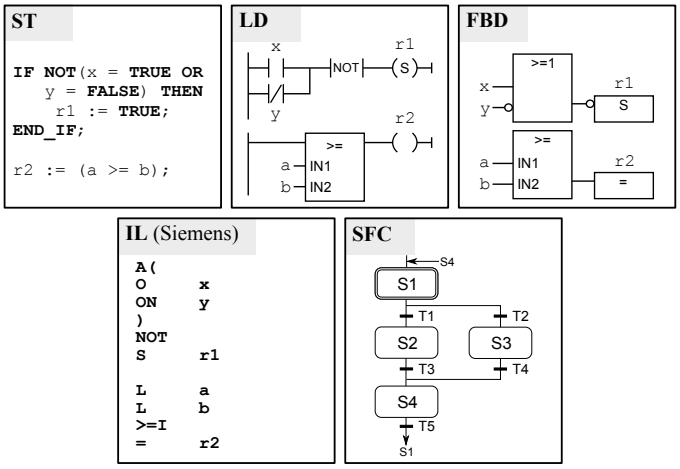


Fig. 1. PLC language examples

From the development point of view, providing a complete parser and a verification model builder is a great effort. Furthermore, the grammars of the PLC languages are notably different, making it difficult to use the same technology stack. For example, the currently used Xtext-based parser is not suitable for the IL language, where the same tokens can be treated as keywords or names depending on the context. On the other hand, the different languages have many common parts, e.g. function and function block declarations, variable declarations. If these should be developed for each language independently, the maintenance of the tool may become difficult.

Instead, in this paper we investigate the possibility of a different approach: *is it possible to use the ST language as a pivot to represent all five standard PLC languages?* If

the translation preserves the properties of the model to be checked, adding this extra translation step (i.e. transformation to ST, then parse and build the verification model) makes no theoretical difference, the pivot language might be considered as a concrete syntax of the underlying verification model. However, as it will be discussed later, the development and maintenance effort needed could be significantly lower.

To answer this question, the relations between the PLC languages have to be investigated. As it might not be possible or practical to translate each language *directly* to ST, the relationship between all languages should be discussed.

B. Related Work

Transformation of PLC programs were already studied previously. For example, Sadolewski translates ST programs into ANSI C [5] and Why [6] for verification purposes. However, in both work the source and target languages are on a similar abstraction level, not necessitating a detailed analysis of the ST language. Sülflow and Drechsler [7] translate IL programs to SystemC representation in order to perform equivalence checking. Here the translation involves a significant change in the abstraction level, requiring more considerations. Furthermore, all of them targeted one single source PLC language. In our work, all the five PLC languages are compared.

The paper is structured as follows. Section II defines our comparison method. Section III discusses the relations between the different IEC 61131 PLC programming languages. Next, Section IV discusses a concrete implementation of these languages, namely the one provided by Siemens. Section V analyzes the results of the paper and draws the conclusions. Finally, Section VI summarizes the paper.

II. Comparison Method

The expressive power of different programming languages is often discussed in computer science. However, the typical answer to these questions for a pair of commonly used languages is that both languages are Turing complete, therefore their expressive power is equivalent.

For our purposes, this is not a useful comparison. Firstly, because these languages are designed for special purposes, therefore they contain many limitations. One of them is the lack of dynamic memory allocation. Together with the lack of recursion [1, Sec. 2.5] and the limited data structures, it is impossible to use more storage than the amount defined explicitly in compilation time. These limitations are parts of the language definition, not caused by implementation or hardware limitations, therefore these languages are *not Turing complete*.

Secondly, when we are looking for a pivot language, it is not enough to know that a certain program can be represented in another language, i.e. for each program in source language S there *exists* an execution equivalent program in language T. It should be known as well, *how* can this translation be performed. Therefore we are interested in a stronger, element-wise emulation relation that determines whether *each "element"*[1] of a language S can be mapped to language T. If this relation holds, then inductively all programs of language S can be translated into language T, in other words language T can emulate language S. This is close to defining a *small-step operational semantics* for language S in language T.

In the following we investigate for each pair of PLC languages if such element-wise mapping relation exists. Note that this relation is transitive, reflexive and asymmetric. We start the investigation with the IEC 61131 version of the languages, as they have a detailed, yet semi-formal description in [1]. Later, we check the differences between the standard and the Siemens variants.

[1] As the PLC languages are significantly different, "element" is understood on a high level (i.e. an element can be a statement, but also a wire junction).

III. Standard Languages

In this section, we discuss the element-wise representation relation for each pair of standard PLC languages. The findings are summarized in Table I. Here "–" denotes that this representation is not possible.

TABLE I. Element-wise mapping between standard languages

from \ to	ST	IL	FBD	LD	SFC
ST	+	+	–	–	–
IL	–	+	–	–	–
FBD	–	+	+	+	–
LD	–	+	+	+	–
SFC	+	+	+	+	+

SFC is based on a specification method called Grafcet, which itself has roots in safety Petri nets. The goal of SFC is to structure the programs, it is not intended to be a generic PLC language. As only certain types of program units can be represented by SFCs, while the other four languages target all of the program unit types, no other language can be represented in SFC. Since it is based on Petri nets, translating the structure of an SFC program to any other language might be problematic, because Petri nets allow non-determinism, while the PLC languages are deterministic. However, determinism is explicitly required by the standard [1, Sec. 2.6.5]. The parts of SFC besides the structure are defined as simple program snippets in other languages and these specific parts can be easily mapped to any other PLC language, assuming that the ambiguities of the standard are first resolved [8].

FBD and **LD** are two similar graphical languages. FBD is composed by signal flow lines and boxes representing built-in and user-defined program units. LD is closer to the electric diagrams, with concepts like power rails, contacts and coils. Despite the differences, IEC 61131 defines LD and FBD in a similar way, with many common elements. The differences [1, Sec. 4.2–4.3] are minor and mainly syntactic. All LD-specific elements (e.g. coils, power rails) can be translated to equivalent FBD elements and vice versa. The wires and flow lines represent data flows, the coils and contacts have corresponding instructions in IL, that is an assembly-like, low-level language. The built-in and user-defined blocks of FBD and LD can be called from IL as well. Therefore each FBD and LD program can be element-wise mapped to IL, in some cases requiring to explicitly introduce new variables that are only implicitly present (as wires) in the FBD and LD programs.

Contrarily, LD and FBD programs cannot be element-wise mapped to ST. The FBD, LD and IL languages support labels and jumps, but ST enforces structured programming, thus jumps are missing from the language [1, Sec. B.3]. Although

it is known that Turing complete programs can be made jump-free by replacing jumps with loops and conditional statements [9], this construction does not fit to our approach of element-wise mapping.

IL has instructions such as `LD` (load value to accumulator) or `ST` (store the accumulator value to the given variable), and the other languages do not provide direct access to the accumulator, this way the element-wise (instruction by instruction) translation to any other PLC language is not possible.

ST is a high-level, structured textual language. Besides providing program structuring elements, such as conditional statements (`IF`, `CASE`) and loops, it also makes the indirect variable access possible. For example, the expression "`array_var[var1]`" is allowed in ST, but not in FBD or LD [1, Sec. 2.4.1.2], therefore the ST to FBD or LD translation is not possible. On the other hand, these expressions are allowed in IL. More precisely, the ST syntax for defining expression is allowed in IL in certain cases. Based on the syntax and semantics definitions of ST and IL, each ST statement can be represented by a list of IL instructions: the corresponding arithmetic operations exist in IL as well, the variable assignments can be performed through `LD` and `ST`, the selection and iteration statements can be represented by labels and jumps, etc.

Based on the above discussion and Table I, ST does not seem to be a pivot language candidate. However, before the final conclusion, the *implementation* of the languages should also be checked for two reasons: (1) the different manufacturers may have differences in their implementation compared to the standard, and (2) the IEC 61131 standard is ambiguous [8], [10] and the vendors might resolve the ambiguities differently. The following section compares a concrete implementation of the five PLC programming languages.

IV. Implementation of the Languages

The IEC 61131 standard does not discuss the implementation details of the languages. Several decisions are left to the vendors, marked as "implementation-dependent" feature or parameter in the standard (e.g. range of certain data types, output values on detected internal errors). Consequently, PLC providers support different variants of the languages. The implementation-dependent details are also important for the behaviour of the programs, thus it is necessary to check these details. Siemens is the PLC provider most used at CERN, therefore we focus on the Siemens variants of the languages in this section. All five languages are supported in Siemens PLCs, however with some differences [11]. Compared to the standard, the differences are significant in some cases, also some languages have ancestors from times before IEC 61131, thus Siemens uses different names for their languages: instead of ST, IL, FBD, LD, SFC the Siemens languages are called SCL, STL, FBD, LAD, SFC/GRAPH, respectively. To avoid the confusion of the readers, we will use the standard language names for the Siemens variants too, with an added apostrophe.

The differences between the standard and Siemens versions of FBD, LD and SFC are subtle and mainly syntactic[2] [11].

[2]For instance, LD' fully implements the standard. The only difference between FBD and FBD' is that the latter does not support the unconditional jumps, but it is easy to represent them as conditional jumps [11].

TABLE II. Element-wise mapping between Siemens languages

from \ to	ST'	IL'	FBD'	LD'	SFC'
ST'	+	+	–	–	–
IL'	–	+	–	–	–
FBD'		+	+	+	
LD'		+	+	+	
SFC'	+	+	+	+	+

Notable differences in syntax and semantics between the standard and the implementation can be observed in the Siemens variants of ST and IL. The following part of the section overviews the differences compared to Table I, see Table II.

As the FBD', LD', and SFC' are equal to the standard versions, the relations between them are valid for the Siemens variants too. ST' and IL' are extended compared to the standard equivalents. Therefore if one of these languages can be mapped to ST or IL, it can be mapped also to the corresponding implementation, and if ST or IL cannot be mapped to one of these languages, IL' or ST' cannot be mapped to the implementation of the same language either. Consequently, the shaded cells of Table II are inherited from Table I.

Due to the limitations of the Siemens development environment, the **FBD'** and **LD'** programs can only be exported if they are translated to IL' first. According to [12], the translation from LD' and FBD' to IL' is always possible. We omit the discussion of transforming LD' and FBD' to ST' or SFC', as they would be practically infeasible.

The Siemens variant of **ST** is significantly extended compared to the standard. It includes labels and jump functions, which invalidates the reasoning of Section III why IL, LD and FBD cannot be represented in ST. Despite the extensions, it is not possible in ST' to directly access the registers, e.g. modifying the contents of the accumulators. Therefore the IL' instruction "`L var1`", transferring the contents of Accumulator 2 to Accumulator 1 and then loading the content of variable `var1` to Accumulator 2 cannot be directly represented in ST'. One can argue that a function containing only the instruction "`L var1`" is meaningless, as its effect will be made invisible when the function returns. However, this example is enough to demonstrate that the element-wise mapping is not possible.

The Siemens variant of **IL** is remarkably different from the standard IL. This is manifested in a different syntax. A short example is the following: the IL program in Listing 1 and the IL' program in Listing 2 give the same outputs to the same inputs, but they use a significantly different syntax and underlying semantics. The behaviour of both code snippets is equivalent to `r:=(a >= b)` in ST. The background of this difference is that the standard defines only one "register", the result variable. The Siemens implementation is closer to the assembly-like languages, using several status bits, registers, accumulators, etc[3]. As the ST' and IL' language definitions are non-formal, it is difficult to argue about the ST' to IL' transformation. However, the Siemens development tool provides this transformation capability, therefore we treat this as possible.

[3]From this point we use the term "register" in a generic way, referring to status bits, accumulators, nesting stack, etc.

```
1 LD a (* RES:=a *)
2 GE b (* RES:=(RES>=b) *)
3 ST r (* r:=RES *)
```

Listing 1. Example IL code

```
1 L a   (* ACC2:=ACC1; ACC1:=a *)
2 L b   (* ACC2:=ACC1; ACC1:=b *)
3 >=I   (* RLO:=(ACC2>=ACC1) *)
4 = r   (* r:=RLO *)
```

Listing 2. Example IL' code

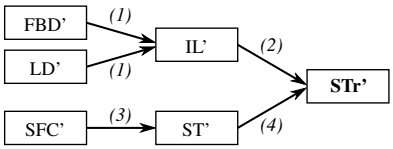


Fig. 2. Unified representation of Siemens PLC languages

V. Analysis and Conclusion

Looking at Table II might lead to the same conclusion for the Siemens implementations of the languages as Table I. However, due to the extensions in the implementations, the gap between IL' and ST' is much smaller than between their standard equivalents. The only difference between them is the possibility to access the registers directly. Therefore ST' can be a pivot language, if it is extended with the *emulation of register access* by using dedicated local variables for verification purposes. We will refer to this format of the programs as *STr'*. As the values of the registers are saved on the stack on each function call, their values are local to each program unit, they can be represented as local temporary variables. Thus the mapping from IL' to STr' can be done instruction by instruction, by explicitly representing the effects of each instruction on the basis of their semantics. For example, the above-mentioned "`L var1`" will be represented as "`ACC2 := ACC1; ACC1 := var1;`", where `ACC1` and `ACC2` are the local variables representing Accumulator 1 and 2. This idea is similar to the SystemC representation used in [7].

Although the FBD' and LD' cannot be directly translated to STr' in practice, it is feasible through IL'. The SFC' programs can directly be mapped to ST', thus to STr' also. The advantage of this method is that one parser and generator to construct the intermediate verification model fits all languages. Only a simpler, text-to-text mapping to STr' has to be developed for each language that is responsible for translating the language-specific parts, element by element.

One could argue that IL' might be a good pivot language without defining any extension or representation convention for the verification. However, STr' is a higher-level language, with a more compact representation (especially the expression description is more compact). The underlying intermediate verification model supports also complex expressions (similarly to the formalism of many model checkers, e.g. nuXmv, UPPAAL), therefore translating a compact ST' expression to a lengthy IL' form is inefficient. Also, in our setting typically ST' codes are verified, therefore using STr' (and not IL') as pivot can provide support for the other languages without any impact on the verification of ST' programs.

Figure 2 summarizes the proposed generic representations of PLC languages for PLCverif. The FBD' and LD' graphical languages can be translated into IL' by the Siemens development environment *(1)*. An instruction-by-instruction transformation from IL' to STr' that makes the effects of the IL' instructions explicit is implemented for the most common instructions *(2)*. The SFC' to ST' translation is relatively simple, it can be implemented using the same principles as the ones used in [2] to represent SFC' directly using the PLCverif intermediate model *(3)*. Finally, ST' is a subset of STr', thus it does not need any further transformation step *(4)*. The STr' code is the input for the verification model generation.

In this paper *interrupts* were not targeted. Certain PLCs may use interrupts, interrupting the execution of the main program. A certain IL' instruction can be atomic, but the corresponding STr' representation, comprising several statements will not be atomic. This might cause concurrency problems and discrepancies between the two representations of the code. However, if this is critical, a locking mechanism can be added to the translation. Although the IEC 61131 standard does not define any locking mechanism, it is defined for the Siemens ST' language via the available system function blocks.

VI. Summary

This paper presented the relations between the different PLC programming languages, both for the standard versions of IEC 61131 and the Siemens implementations. For our practical goals, i.e. to extend PLCverif to support all five Siemens variants of the PLC languages, a good pivot language candidate was found: STr' that is the Siemens ST' language emulating register access with variable access for verification purposes. Using STr', PLCverif can efficiently support the verification of low-level languages (IL', FBD', LD'), without modifying the core workflow or decreasing the verification performance of the programs written in ST' language.

References

[1] *IEC 61131-3:2003 Programmable controllers – Part 3: Programming languages*, IEC Std., 2003.

[2] B. Fernández *et al.*, "Applying model checking to industrial-sized PLC programs," *IEEE Transactions on Industrial Informatics*, vol. 11, no. 6, pp. 1400–1410, 2015.

[3] D. Darvas, B. Fernández, and E. Blanco, "PLCverif: A tool to verify PLC programs based on model checking techniques," in *Proc. of the 15th Int. Conf. on Accelerator & Large Experimental Physics Control Systems*, 2015.

[4] B. Fernández, D. Darvas, J.-C. Tournier, E. Blanco, and V. M. González, "Bringing automated model checking to PLC program development – A CERN case study," in *Proc. of the 12th Int. Workshop on Discrete Event Systems*. IFAC, 2014, pp. 394–399.

[5] J. Sadolewski, "Conversion of ST control programs to ANSI C for verification purposes," *e-Informatica*, vol. 5, no. 1, pp. 65–76, 2011.

[6] ——, "Automated conversion of ST control programs to Why for verification purposes," in *Proc. of the Federated Conf. on Computer Science and Information Systems*. IEEE, 2011, pp. 849–854.

[7] A. Sülflow and R. Drechsler, "Verification of PLC programs using formal proof techniques," in *Formal Methods for Automation and Safety in Railway and Automotive Systems*. L'Harmattan, 2008, pp. 43–50.

[8] N. Bauer, R. Huuck, B. Lukoschus, and S. Engell, "A unifying semantics for sequential function charts," in *Integration of Software Specification Techniques for Applications in Engineering*, ser. Lecture Notes in Computer Science. Springer, 2004, vol. 3147, pp. 400–418.

[9] C. Böhm and G. Jacopini, "Flow diagrams, turing machines and languages with only two formation rules," *Communications of the ACM*, vol. 9, no. 5, pp. 366–371, 1966.

[10] M. de Sousa, "Proposed corrections to the IEC 61131-3 standard," *Computer Standards & Interfaces*, vol. 32, no. 5-6, pp. 312–320, 2010.

[11] Siemens, "Standards compliance according to IEC 61131-3," 2011, http://support.automation.siemens.com/WW/view/en/50204938.

[12] ——, *SIMATIC Ladder Logic (LAD) for S7-300 and S7-400 Programming*, 1996, C79000-G7076-C504-02.

Towards Efficient CEGAR-Based Reachability Analysis of Timed Automata

Rebeka Farkas, András Vörös
Budapest University of Technology and Economics
Department of Measurement and Information Systems
Budapest, Hungary
Email: rebeka.farkas@inf.mit.bme.hu, vori@mit.bme.hu

***Abstract*—The verification of safety-critical real-time systems can find design problems at various phases of the development or prove the correctness. However, the computationally intensive nature of formal methods often prevents the successful verification. Abstraction is a widely used technique to construct simple and easy to verify models, while counterexample guided abstraction refinement (CEGAR) is an algorithm to find the proper abstraction iteratively. In this work we extend the CEGAR framework with a new refinement strategy yielding better approximations of the system. A prototype implementation is provided to prove the applicability of our approach.**

I. Introduction

It is important to be able to model and verify timed behavior of real-time safety-critical systems. One of the most common timed formalisms is the timed automaton that extends the finite automaton formalism with real-valued variables – called clock variables – representing the elapse of time.

A timed automaton can represent two aspects of the behavior. The discrete behavior is represented by locations and discrete variables with finite sets of possible values. The time-dependent behavior is represented by the clock variables, with a continuous domain.

A timed automaton can take two kinds of steps, called transitions: discrete and timed. A discrete transition changes the automaton's current location and the values of the discrete variables. In addition, it can also reset clock variables, which means it can set their value to 0. Time transitions represent the elapse of time by increasing the value of each clock variable by the same amount. They can not modify the values of discrete variables. Transitions can be restricted by guards and invariants.

In case of real-time safety-critical systems, correctness is critical, thus formal analysis by applying model checking techniques is desirable. The goal of model checking is to prove that the system represented by the model satisfies a certain property, described by some kind of logical formula. Our research is limited to reachability analysis where the verification examines if a given set of (error) states is reachable in the model. Reachability criterion defines the states of interest.

Many algorithms are known for model checking timed systems, the one which defines an efficient abstract domain to handle timed behaviors is presented in [1]. The abstract domain is called *zone*, and it represents a set of reachable valuations of the clock variables. The reachability problem is decided by traversing the so-called *zone graph* which is a finite representation (abstraction) of the continuous state space.

Model checking faces the so-called state space explosion problem – that is, the statespace to be traversed can be exponential in the size of the system. It is especially true for timed systems: complex timing relations can necessiate a huge number of zones to represent the timed behaviors. A possible solution is using abstraction: a less detailed system description is desired which can hide unimportant parts of the behaviors providing less complex system representations.

The idea of counterexample guided abstraction refinement (CEGAR) [2] is to apply model checking to this simpler system, and then examine the results on the original one. If the analysis shows, that the results are not applicable to the original system, some of the hidden parts have to be re-introduced to the representation of the system – i.e., the abstract system has to be refined. This technique has been successfully applied to verify many different formalisms.

Several approaches have been proposed applying CEGAR on timed automata. In [3] the abstraction is applied on the locations of the automaton. In [4] the abstraction of a timed automaton is an untimed automaton. In [5], [6], and [7] abstraction is applied on the variables of the automaton.

Our goal is to develop an efficient model checking algorithm applying the CEGAR-approach to timed systems. The above-mentioned algorithms modified the timed automaton itself: our new algorithm focuses on the direct manipulation of the reachability graph, represented as a zone graph, which can yield the potential to gain finer abstractions.

The paper is organized as follows. Section II provides some definitions and basic knowledge about timed automata and CEGAR. In Section III our approach is explained, and a simple implementation is described. Section IV gives some final remarks.

II. Background

In this section we define the important aspects of timed automata and briefly explain the relevant parts of the reachability algorithm presented in [1]. We present the verification of Fischer's protocol as an example. CEGAR is also explained at a high level.

A. Basic Definitions

A *valuation* $v(\mathcal{C})$ assigns a non-negative real value to each clock variable $c \in \mathcal{C}$, where $\mathcal{C}$ denotes the set of clock variables.

A *clock constraint* is a conjunctive formula of atomic constraints of the form $x \sim n$ or $x - y \sim n$ (*difference constraint*), where $x, y \in \mathcal{C}$ are clock variables, $\sim \in \{\leq, <, =, >, \geq\}$ and $n \in \mathbb{N}$. $\mathcal{B}(\mathcal{C})$ represents the set of clock constraints.

A *timed automaton* $\mathcal{A}$ is a tuple $\langle L, l_0, E, I \rangle$ where L is the set of locations, $l_0 \in L$ is the initial location, $E \subseteq L \times \mathcal{B}(\mathcal{C}) \times 2^{\mathcal{C}} \times L$ is the set of edges and, $I : L \to \mathcal{B}(\mathcal{C})$ assigns invariants to locations. Invariants can be used to ensure the progress of time in the model.

A state of $\mathcal{A}$ is a pair $\langle l, v \rangle$ where $l \in L$ is a location and v is the current valuation satisfying $I(l)$. In the initial state $\langle l_0, v_0 \rangle$ v_0 assigns 0 to each clock variable.

Two kinds of operations are defined. The state $\langle l, v \rangle$ has a *discrete transition* to $\langle l', v' \rangle$ if there is an edge $e(l, g, r, l') \in E$ in the automaton such that v satisfies g, v' assigns 0 to any $c \in r$ and assigns $v(c)$ otherwise and v' satisfies $I(l')$. The state $\langle l, v \rangle$ has a *time transition* to $\langle l, v' \rangle$ if v' assigns $v(c) + d$ for some non-negative d to each $c \in \mathcal{C}$ and v' satisfies $I(l)$.

B. Reachability Analysis

A *zone* is a set of nonnegative clock valuations satisfying a set of clock constraints. The set of all valuations reachable from a zone z by time transitions is denoted by $z^{\uparrow}$.

A *zone graph* is a finite graph consisting of $\langle l, z \rangle$ pairs as nodes, where $l \in L$ refers to some location of a timed automaton and z is a zone. Therefore, a node denotes a set of states. Edges between nodes denote transitions.

The construction of the graph starts with the initial node $\langle l_0, z_0 \rangle$, where l_0 is the initial location and z_0 contains the valuations reachable in the initial location by time transition. Next, for each outgoing edge e of the initial location (in the automaton) a new node $\langle l, z \rangle$ is created (in the zone graph) with an edge $\langle l_0, z_0 \rangle \to \langle l, z \rangle$, where $\langle l, z \rangle$ contains the states to which the states in $\langle l_0, z_0 \rangle$ have a discrete transition through e. Afterwards z is replaced by $z^{\uparrow}$. The procedure is repeated on every newly introduced node of the zone graph. If the states defined by a newly introduced node $\langle l, z \rangle$ are all contained in an already existing node $\langle l, z' \rangle$, $\langle l, z \rangle$ can be removed, and the incoming edge should be redirected to $\langle l, z' \rangle$. Unfortunately the described graph can possibly be infinite.

A concept called *normalization* is introduced in [1]. Let $k(c)$ denote the greatest value to which clock c is compared. For any valuations v such that $v(c) > k(c)$ for some c, each constraint in the form $c > n$ is satisfied, and each constraint in the form $c = n$ or $c < n$ is unsatisfied, thus the interval $(k(c), \infty)$ can be used as one abstract value for c. Normalization is applied on $z^{\uparrow}$ before inclusion is checked. Using normalization the zone graph is finite, but unreachable states may appear.

The operation *split* [1] is introduced to eliminate such states. Instead of normalizing the complete zone, it is first split along the difference constraints, then each subzone is normalized, and finally the initially satisfied constraints are reapplied to each zone. If the result is a set of zones, then multiple new nodes have to be introduced to the zone graph. Applying split results in a zone graph, that is a correct and finite representation of the state space.

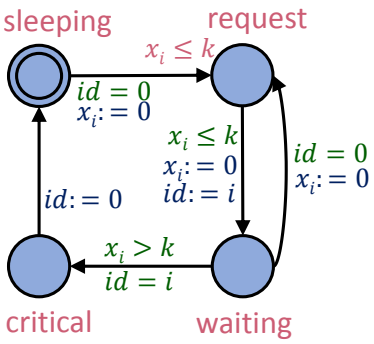


Fig. 1. Fischer's protocol

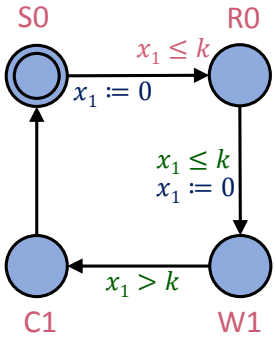


Fig. 2. Timed automaton

Example: Fischer's protocol assures mutual exclusion by bounding the execution times of the instructions. It can be applied to a number of processes accessing a shared variable. Fig. 1 shows the operation of a process. The location *critical* indicates that the process is in the critical section. The value of the shared variable id ranges between 0 and n, where n denotes the number of processes. The model also contains a clock variable x_i for each process where $i \in \{1 \ldots n\}$ denotes the identifier of the process. The constant k is a parameter of the automaton.

The mutual exclusion property would suggest that at any given time at most one of the processes is in the *critical* location. In order to check the given property we must construct a timed automaton that models the operation of a given number of processes.

As our definition of timed automaton only allows clock variables in the system, everything else must be encoded in the location. To demonstrate, Fig. 2 shows the reachable locations of the product automaton of Fischer's protocol where $n = 1$. The names of the locations refer to the original locations of the process, the number denotes the value of the variable id.

C. CEGAR

The CEGAR approach introduced in [2] makes abstraction refinement a key part of model checking. The idea is illustrated on Fig. 3.

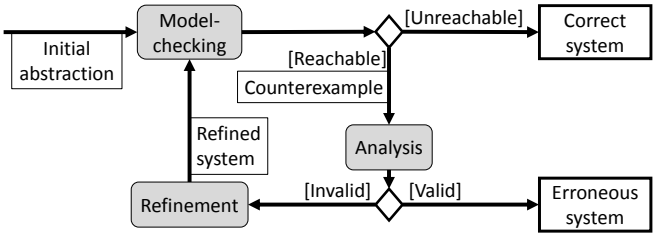


Fig. 3. Counterexample guided abstraction refinement

First, an abstract system is constructed. The key idea behind abstraction is that the state space of the abstract system overapproximates that of the original one. Model checking is performed on this abstract model. If the target state is unreachable in the abstract model, it is unreachable in the original model as well. Otherwise the model-checker produces a counterexample – a run where the system reaches the target state. In our case the counterexample is a sequence of transitions – i.e., a trace. Overapproximation brings such behaviors to the system that are not feasible in the original one. Because of this, the counterexample may not be a valid trace in the real system, so it has to be investigated. If it turns out to be a feasible counterexample, the target state is reachable. Otherwise the abstract system has to be refined. The goal of the refinement is to modify the abstract system so that it remains an abstraction of the original one, but the spurious counterexample is eliminated. Model checking is performed on the refined system, and the CEGAR-loop starts over.

The algorithm terminates when no more counterexample is found or when a feasible trace is given leading to the erroneous state.

III. Applying CEGAR to the Zone Graph

In this section we explain our approach of applying CEGAR to the timed automaton. Some details of our implementation are also discussed.

A. Phases of CEGAR

Our algorithm is explained in this section. To ease presentation, we illustrate the algorithm on an example. The timed automaton is on Fig. 2 and the value of the parameter k is 2. The property to check is whether the automaton can reach the critical section.

1) Initial Abstraction: The first step of the CEGAR-approach is to construct an initial abstraction, which is an over-approximation of the system's state space. In our algorithm, the state space is represented by the zone graph. Instead of constructing the zone graph of the system an abstract simpler representation is constructed from the timed automaton.

Erroneous states are represented by (erroneous) locations, so we decided not to apply abstraction on them. However, the zones are overapproximated – the initial assumption is that every valuation is reachable at every location. This means that the initial abstraction of the zone graph will contain a node $\langle l, z_\infty \rangle$ for each location l, where z_∞ is the zone defined by the constraint set $\{c \geq 0 \mid c \in \mathcal{C}\}$.

Edges of the abstract zone graph can also be derived from the timed automaton itself. If there is no edge in the automaton leading from location l to l' there can not be a corresponding edge $\langle l, z \rangle \rightarrow \langle l', z' \rangle$ in the (concrete) zone graph regardless of z and z'. Thus, there should not be an edge from $\langle l, z_\infty \rangle$ to $\langle l, z'_\infty \rangle$ in the abstract zone graph either. All other edges are represented in the initial abstraction.

This results in a graph containing locations (extended with the zone z_∞) as nodes, and edges of the automaton (without guard and reset statements) – an untimed zone graph, derived completely from the automaton as the real zone graph is unknown. The initial abstraction derived from the example timed automaton can be seen on Fig. 4.

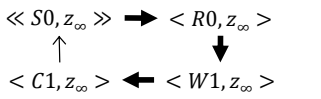


Fig. 4. The abstract zone graph of the automaton

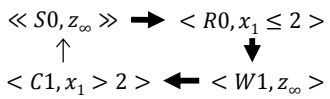


Fig. 5. The refined graph

This will be the model on which we apply model checking.

2) Model Checking: During the reachability analysis only the counterexample traces will be refined in the zone graph. Thus, model checking becomes a pathfinding problem in the current abstraction of the zone graph in each iteration. Either we prove the target state to be unreachable or a new path is found from the initial node to the target node.

The result of pathfinding in the graph on Fig. 4 is denoted by bold arrows.

3) Counterexample Analysis and Refinement: Analyzing the counterexample in the original system and refining the abstract representation are two distinct steps of CEGAR, but in our approach they are performed together. The goal of refinement is to eliminate the unreachable states from the abstract representation. Refinement is applied by replacing the abstract zones in the counterexample trace with refined zones containing only reachable states.

In the first iteration, no nodes of the abstract graph has ever been refined, so refinement starts from the node that belongs to the initial location where the refined zone is calculated from the initial valuation. In case of the later iterations the first few nodes of the trace will already be refined, so the refinement can start from the first abstract node. The reachable zone should be calculated from the last refined zone, considering the guards and the reset as described in [1].

Of course, as discussed in Section II-B sometimes the result of the refinement is more than one zone. In this case the node in the graph (and the edge pointing to it) is replicated, and one of the refined zones are assigned to each resulting node. The refinement can be continued from any of these nodes – the path branches. All of these branches should be analyzed (refined) one by one.

It is also advised to reuse zones already refined. Suppose at one point of the algorithm the zone z_∞ of the node $\langle l, z_\infty \rangle$ is refined to z, and z is a subzone of a zone z' in a node $\langle l, z' \rangle$ (both nodes contain the same location l). In this case any state that is reachable from $\langle l, z \rangle$ is also reachable from $\langle l, z' \rangle$, thus any edge leading to $\langle l, z \rangle$ is redirected to $\langle l, z' \rangle$, and $\langle l, z \rangle$ is removed. After that the analysis of the path can continue from that $\langle l, z' \rangle$.

If the erroneous location is reachable through this path, the procedure finds it, and the CEGAR algorithm terminates. Otherwise, at some point a guard or a target invariant is not satisfied – the transition is not enabled. The corresponding edge is removed and the analysis of the path terminates.

Let us consider the example. Refining the path on Fig. 4 is performed as follows. Refinements starts with the initial node $\langle S0, z_\infty \rangle$. First, we must consider the edge $\langle C1, z_\infty \rangle \rightarrow \langle S0, z_\infty \rangle$. The refinement will eliminate any state that is unreachable in the initial node of the zone graph, but there might be another node in the real zone graph with the location $S0$, so we duplicate the node before the refinement and the edge $\langle C1, z_\infty \rangle \rightarrow \langle S0, z_\infty \rangle$ will point to the duplicate. The value of x_1 is 0 in the initial state. Before the discrete transition occurs, any delay is enabled (as there is no location invariant on $S0$), so x_1 can take any non negative value. Thus $\{x_1 > 0\} = z_\infty$ is the zone assigned to the initial location. Since it is contained in the existing $\langle S0, z_\infty \rangle$ (the duplicate), $\langle S0, x_1 > 0 \rangle$ (the refined node) can be removed and the analysis of the path continues from the remaining node $\langle S0, z_\infty \rangle$.

The next node to refine is $\langle R0, z_\infty \rangle$. The transition from $\langle S0, z_\infty \rangle$ resets x_1, so its initial value in location $R0$ is 0. The invariant of the location limits the maximum value of x_1, hence the maximum value of a time transition at location $R0$ is 2. Thus the reachable zone in $R0$ satisfies $x_1 \leq 2$. The refinement of the trace continues, and $C1$ turns out to be reachable. The refined zone graph is depicted on Fig 5.

B. Implementation

A simple implementation of the method is explained in this section. Please note that this is still an on-going research, and our current proof-of-concept implementation is far from optimal.

1) Data Structure: The zone graph is represented by an auxiliary graph that can be formally defined as a tuple $\langle N_A, N_R, E^\uparrow, E^\downarrow \rangle$ where $N_A \subseteq L \times \{z_\infty\}$ is the set of abstract nodes, $N_R \subseteq L \times \mathcal{B}(\mathcal{C})$ is the set of refined nodes, $E^\uparrow \subseteq (N_A \times N_A) \cup (N_R \times N_R)$ is the set of upward edges, and $E^\downarrow \subseteq N \times N$ where N denotes $N_A \cup N_R$ is the set of downward edges. The sets $E^\uparrow$ and $E^\downarrow$ are disjoint. $T^\downarrow = (N, E^\downarrow)$ is a tree. The depth of a node n in T is denoted by $d(n)$.

Nodes are built from a location and a zone like in the zone graph but in this case nodes are distinguished by their trace reaching them from the initial node. This means the graph can contain multiple nodes with the same zone and the same location, if the represented states can be reached through different traces. The root of T is the initial node of the (abstract) zone graph. A downward edge e points from node n to n' if n' can be reached from n in one step in the zone graph. In this case $d(n') = d(n) + 1$.

Upward edges are used to collapse infinite traces of the representation, when the states are explored in former iterations. An upward edge from a node n to a node n' where $d(n') < d(n)$ means that the states represented by n are a subset of the states represented by n', thus it is unnecessary to keep searching for a counterexample from n, because if there exists one, a shorter one will exist from n'. Searching for new traces is only continued on nodes without an upward edge. This way, the graph can be kept finite.

Refined nodes appear in the refinement phase. Upward edges can point from abstract to abstract, or from refined to refined node.

2) Applying our Algorithm to the Graph Structure: In our implementation model checking is performed by breadth-first search (BFS). The graph is built until we reach a node containing the target location at some depth (or the location turns out to be unreachable). The trace can be computed by stepping upward on the downward egdes to the root of T.

Before refining a node n, we take care of the upward edges pointing to it. Since the node is about to be refined no upward edge from an abstract zone should point to it. Thus, if there is an edge $n' \rightarrow n \in E^\uparrow$, we remove it, and instead continue BFS from n', until the current depth of T is reached. It can be proven that this will not introduce new counterexamples.

Refinement is performed by replacing abstract nodes with refined ones. Refined zones are calculated as described above. If at some point a node is replicated (because of the split operation), then the subtree should be copied as well. New paths are introduced in the new subtrees, that have to be analyzed later.

If a zone z is a subzone of z' where $d(\langle l, z \rangle) > d(\langle l, z' \rangle)$ we introduce an upward edge $\langle l, z \rangle \rightarrow \langle l, z' \rangle$, and terminate the analysis of the current trace, as it can be proven that it will turn out to be invalid.

IV. Conclusion

This paper introduced a new CEGAR-based reachability analysis of timed automata. Unlike the similar existing approaches, where the refinement phase focuses on the complete automaton, the key idea of our approach is to refine the zone graph incrementally. A prototype implementation is also introduced.

References

[1] J. Bengtsson and W. Yi, "Timed automata: Semantics, algorithms and tools," in *Lectures on Concurrency and Petri Nets*, ser. LNCS. Springer Berlin Heidelberg, 2004, vol. 3098, pp. 87–124.

[2] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-guided abstraction refinement for symbolic model checking," *Journal of the ACM (JACM)*, vol. 50, no. 5, pp. 752–794, 2003.

[3] S. Kemper and A. Platzer, "SAT-based abstraction refinement for real-time systems," *Electronic Notes in Theoretical Computer Science*, vol. 182, pp. 107–122, 2007.

[4] T. Nagaoka, K. Okano, and S. Kusumoto, "An abstraction refinement technique for timed automata based on counterexample-guided abstraction refinement loop," *IEICE Transactions*, vol. 93-D, no. 5, pp. 994–1005, 2010.

[5] H. Dierks, S. Kupferschmid, and K. G. Larsen, "Automatic abstraction refinement for timed automata," in *Formal Modeling and Analysis of Timed Systems, FORMATS'07*, ser. LNCS. Springer, 2007, vol. 4763, pp. 114–129.

[6] F. He, H. Zhu, W. N. N. Hung, X. Song, and M. Gu, "Compositional abstraction refinement for timed systems," in *Theoretical Aspects of Software Engineering*. IEEE Computer Society, 2010, pp. 168–176.

[7] K. Okano, B. Bordbar, and T. Nagaoka, "Clock number reduction abstraction on CEGAR loop approach to timed automaton," in *Second International Conference on Networking and Computing, ICNC 2011*. IEEE Computer Society, 2011, pp. 235–241.

Partial Volume Artifact in MITS Reconstructed Digital Chest Tomosynthesis

Dániel Hadházi, Gábor Horváth
Department of Measurement and Information Systems
Budapest University of Technology and Economics
Budapest, Hungary
Email: {hadhazi, horvath}@mit.bme.hu

***Abstract*—Matrix Inversion Tomosynthesis (MITS) is a linear shift invariant MIMO system that deconvolves the X-ray projections that are acquired in tomosynthesis arrangement. As MITS theoretically models the examined volume by finite, zero thin slices, it has a special effect called partial volume artifact. This paper describes the basics of the MITS reconstruction and the theory of the analyzing of the reconstruction using Modulation Transfer Function (MTF) in detail. The dependency of the partial volume artifact on the number of reconstructed slices is analyzed using MTF and illustrated by slice images. Based on our results reconstructing more than 350 slices is suggested in the case of imaging a 250 mm thick volume.**

Keywords—chest tomosynthesis; MITS; MTF; partial volume artifact

I. Introduction

Digital chest tomosynthesis is a relatively new imaging modality that computes reconstructed slice images from a set of low-dose X-ray projections acquired over a limited angle range by a flat panel detector [1]. Theoretically this modality is equivalent to a limited angle Cone-beam CT. Tomosynthesis was originally developed for breast screening, however nowadays it is starting to be applied in clinical chest screening purposes, too. In this article linear chest tomosynthesis is examined, where the basic arrangement is illustrated by Fig. 1.

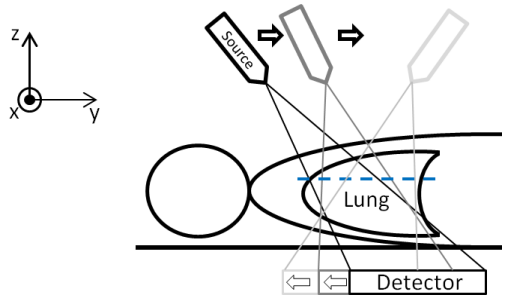


Fig. 1. Schematic arrangement of linear chest tomosynthesis.

In linear chest tomosynthesis arrangement the X-ray source and the flat panel detector move continuously in parallel along the *y* axis in opposite directions. During the motion about 50 projection images are acquired from different positions. According to the figure, the acquired images are located in the *x*-*y* plane, where the rows of the images are parallel with the *x* axis, and the columns of the images are parallel with the *y* axis. From these projections the coronal slices of the 3D examined volume (in our case mainly the lung) are reconstructed (one of them is illustrated by the blue dashed line in the figure). This paper focuses on the partial volume artifact in the case of using Matrix Inversion Tomosynthesis (MITS) algorithm [2] for reconstruction. MITS, which is described in the second section in detail, deconvolves the projections of the coronal slices from the X-ray projections. In the third section the transfer function based analysis of the reconstruction is presented, in section IV the partial volume problem and a possible technique of its treatment is described. Finally in section V conclusions are discussed, and the required number of slices to avoid the artifact caused by the partial volume problem in the case of chest tomosynthesis reconstructed by MITS is determined.

This work was supported in part by ARTEMIS Joint Undertaking and in part by the Hungarian Ministry of National Development within the framework of the Reconfigurable ROS-based Resilient Reasoning Robotic Cooperating Systems Project.

II. Matrix Inversion Tomosynthesis

In the case of tomosynthesis, the projection images can be modelled as the sum of shifted projections of infinitesimally thin slices of the examined volume [2]. In the special case of linear tomosynthesis arrangement, shifting is always along the *y* axis, and the amount of shifting can be expressed analytically as a function of the geometry of the projections' scanning arrangement (illustrated by Fig. 2.) that is detailed later.

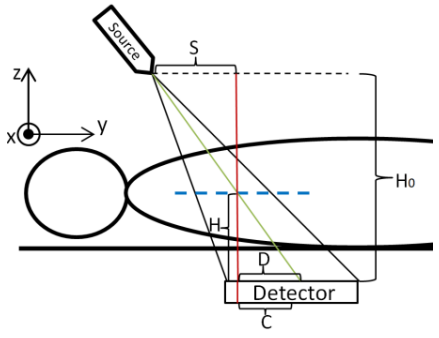


Fig. 2. Projection geometry of linear chest tomosynthesis

In the figure the trajectory of the X-ray beam source is denoted by the black dashed line, and the examined slice is marked by the blue dashed line (*H* denotes its distance from the detector). The red line marks the *x*-*z* plane located at *y*=0, the green

line models the projection beam of the center (x=y=0) of the examined slice (its intersection with the detector's surface is at x=0, y=D). The distance between the detector's surface and the beam's trajectory is denoted by H_0, and the position of the beam source is $(x, y) = (0, S)$, and the center of the detector is at $(x, y) = (0, C)$. The projection of the examined slice is shifted parallel to the y axis, the displacement compared to the situation where the y position of the beam source is 0, and the center of the detector is at $y = 0$, is:

$$D - C = \frac{H \cdot S}{H - H_0} - C \tag{1}$$

Due to the linear tomosynthesis arrangement the shift along the x axis is always zero, so for different x values the intensity profiles can be processed independently of each other as one-dimensional functions of y. Hereafter we will call projections the continuous intensity profiles of the 2D X-ray projection images, that located parallel to the y axis at fixed x positions. These projections are:

$$\begin{aligned} i^{(h)} &= \sum_k \left(r^{(k)} * f_{(k,h)} \right) \\ f_{(k,h)}(x) &= \delta\left(x - D_{(k,h)} \right) \end{aligned} \tag{2}$$

where $i^{(h)}$ denotes the h-th X-ray projection, $r^{(k)}$ denotes the central projection of the k-th modelled slice and $*$ denotes convolution. Central projection means the projection where both the beam source and the detector are at $y = 0$ position, while in modelled slices we mention the non-empty slices (every other slices of the volume presumed empty). $f_{(k,h)}$ is the weight function between the k-th modelled slice and the h-th acquired projection, while the $D_{(k,h)}$ is defined by (1).

The goal of the MITS algorithm [2] is to deblur the conventional Shift And Add (SAA) [3] reconstructed slices. The SAA algorithm is equivalent to the conventional back projection (in the case of linear tomosynthesis), apart from that the outputs of the SAA reconstruction are the projections of the reconstructed slices. The SAA can be modeled as a MIMO linear shift invariant system:

$$\begin{aligned} t^{(h)} &= \sum_h \left(i^{(h)} * f'_{(k,h)} \right) \\ f'_{(k,h)}(x) &= \delta\left(x + D_{(k,h)} \right) \end{aligned} \tag{3}$$

where $t^{(h)}$ denotes the h-th SAA reconstructed slice projection. Based on (2) and (3) the SAA reconstructed slice projections can be expressed as the function of the central projections of the modeled slices:

$$t_j^{(k)} = \sum_h \left(\sum_l r_j^{(l)} * f_{(l,h)} \right) * f'_{(k,h)} \tag{4}$$

The MITS algorithm calculates the estimation of the projections of the modeled slices by deblurring the SAA reconstructed slice projections. As MITS is designed to invert a linear shift invariant MIMO system, it can be derived more easily in spectral domain.

By applying Fourier transform to the second rows of (3) and (2), it is obtained that:

$$FT_\omega\left\{ f_{(k,h)}(x) \right\} = \exp\left(-2\pi j \cdot \omega \cdot D_{(k,h)} \right) = \overline{FT_\omega\left\{ f'_{(k,h)}(x) \right\}} \tag{5}$$

where $\bar{x}$ denotes the complex conjugate of x, and $FT_\omega\{\ \}$ denotes the operator of 1D continuous Fourier transform at ω frequency, j denotes the imaginary unit. Based on this observation (2) and (3) are equivalent to the following:

$$\begin{aligned} \hat{\mathbf{i}}(\omega) &= \mathbf{F}(\omega) \cdot \hat{\mathbf{r}}(\omega) \\ \hat{\mathbf{t}}(\omega) &= \mathbf{F}(\omega)^* \cdot \hat{\mathbf{i}}(\omega) \end{aligned} \tag{6}$$

where $\hat{\mathbf{i}}(\omega)$ denotes a column vector containing the spectral components of the input projections at ω frequency. Similarly $\hat{\mathbf{r}}(\omega)$ is the column vector of the spectrum of the projection of the modeled slices and $\hat{\mathbf{t}}(\omega)$ is the column vector of the spectrum of the SAA reconstructed slice projections at ω frequency. $\mathbf{F}(\omega)$ denotes the projection matrix, where its elements are defined by:

$$\mathbf{F}(\omega)_{(i,j)} = FT_\omega\left\{ f_{(j,i)} \right\} \tag{7}$$

where $\mathbf{A}_{(i,j)}$ denotes the (i, j)-th element of the $\mathbf{A}$ matrix.

By substituting the first equation into the second equation of (6), the spectral representation of (4) is derived:

$$\hat{\mathbf{t}}(\omega) = \mathbf{F}(\omega)^* \cdot \mathbf{F}(\omega) \cdot \hat{\mathbf{r}}(\omega) \tag{8}$$

The SAA reconstructed slice projections can be expressed as a function of the slice projections (based on (6)):

$$\mathbf{F}(\omega)^* \cdot \hat{\mathbf{i}}(\omega) = \mathbf{F}(\omega)^* \cdot \mathbf{F}(\omega) \cdot \hat{\mathbf{r}}(\omega) \tag{9}$$

Therefore deblurring the SAA reconstructed slice projections is equivalent to inverting the system described by the first row of (6):

$$\hat{\mathbf{r}}_j(\omega) = \mathbf{F}(\omega)^\dagger \cdot \hat{\mathbf{i}}_j(\omega) \tag{10}$$

where $\mathbf{F}(\omega)^\dagger$ denotes the pseudo inverse of the projection matrix, $\hat{\mathbf{r}}_j(\omega)$ denotes the estimation of $\hat{\mathbf{r}}_j(\omega)$. From this point of view the MITS can be interpreted as a Maximum Likelihood estimation, formally:

$$\hat{\mathbf{r}}_j(\omega) = \arg\max_{\mathbf{x}} \left\{ \Pr\left(\hat{\mathbf{i}}(\omega) \middle| \mathbf{x} \right) \right\} \tag{11}$$

where $\Pr(\mathbf{y}|\mathbf{x})$ is defined by a multivariate Gaussian distribution, with the expected value of $\mathbf{F}(\omega)\cdot\mathbf{x}$, and the covariance matrix of $\sigma^2\cdot\mathbf{I}$, where $\sigma\in\mathbb{R}^+$.

The theory described previously is valid if and only if special conditions are fulfilled. The three most significant conditions are: (*) $FT_\omega\{\cdot\}$ denotes continuous Fourier transform. In digital signal processing the spectrum can be calculated by discrete Fourier transform (DFT). The problem of under-sampling can be avoided by using a detector for image acquisition with sufficiently high resolution. The distortion caused by spectral leakage can be reduced by sufficiently modifying the projections before calculating the reconstruction. The second (**) assumption is that $\mathbf{F}(\omega)$ should be well-, or over-determined, otherwise numerical problems can significantly decrease the quality of the MITS reconstructed slices. As it is shown in [4] - mainly in low frequencies - this condition is always violated and causes numerical sensitivity of the reconstruction. The third (***) assumption is connected to (2) and (4). These equations are adequately modelling the reality if and only if the volume modelled by an infinite number of infinitesimal thin slices ($r^{(k)}$-s). However this can't be realized which causes artifact. As a similar effect in PET and CT modalities is called partial volume effect [7] we will call it as partial volume artifact of tomosynthesis. This paper focuses on this problem, while the other two problems will be examined in forthcoming publications.

III. Transfer Functions of the Reconstruction

As it is given by (10), MITS is considered as a shift invariant linear multiple input–multiple output (MIMO) system. Therefore MITS is characterized in the spatial frequency domain by its transfer function. The transfer function is determined from the reconstruction of the noise-free simulated projections of a so called wire phantom. This phantom contains an infinitesimally thin wire, whose points are defined along a 1D line:

$$\begin{aligned} y &= x\cdot\tan(\theta)\\ z &= x\cdot\tan(\alpha)\cdot\sqrt{\tan^2(\theta)+1}+z0 \end{aligned} \tag{12}$$

where θ denotes the angle between the perpendicular projection of the wire to the $z=0$ plane, and the $z=y=0$ lines. α denotes the angle between the wire and the $z=0$ plane, $z0$ denotes the offset of the wire along the z axis. The value of $z0$, α and θ define the part of the 3D space which reconstruction can be examined by this analysis. The position of the projection of the wire along the x axis is (independently of the positions of the beam source and the center of the detector along the y direction):

$$p(z)=\frac{H_0-z}{H_0}\cdot\frac{z-z0}{\tan(\alpha)\cdot\sqrt{\tan^2(\theta)+1}} \tag{13}$$

Let $\mathbf{R}^{(i)}$ denote the projection of the i-th reconstructed slice calculated by MITS from the projections of the wire. Due to invertibility of $p(l)$ $\mathbf{R}^{(i)}_{(:,p(l))}$ is the weight function between the projection of the plane of the examined volume located at $z=l$ and the *i*-th reconstructed slice. From this reconstruction the transfer functions of the MITS reconstruction can be calculated:

$$\begin{aligned} \mathbf{O}^{(i)}_{(\omega,l)} &= DFT_\omega\left\{\mathbf{R}^{(i)}_{(:,p(l))}\right\}\\ \mathbf{M}^{(i)}_{(\omega,l)} &= \left|\mathbf{O}^{(i)}_{(\omega,l)}\right| \end{aligned} \tag{14}$$

where $\mathbf{O}^{(i)}_{(:,l)}$ denotes the optical transfer function (OTF) of the reconstruction of the slice located at $z=l$, while $\mathbf{M}$ denotes the modulation transfer function (MTF) of the reconstruction. Fig. 3. illustrates $\mathbf{M}^{(i)}$ as an intensity image.

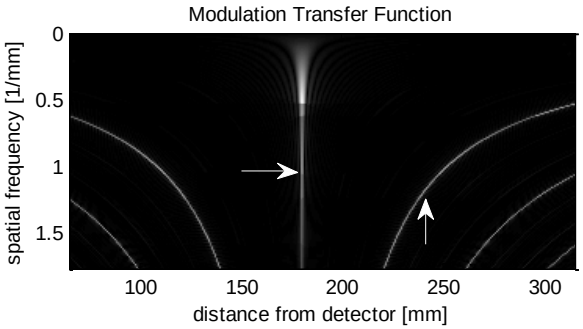


Fig. 3. MTF of a MITS reconstructed slice. The horizontal arrow points to the main lobe, which determine the height of structures reconstructed as in-plane signal. The vertical arrow points to leakage from distant planes.

IV. Partial Volume Artifact

According to [5] the number of the reconstructed slices should not be more than the number of the projections, however our previously published results confirmed that the number of the reconstructed slices can be increased without decreasing the quality of the reconstruction [6]. In the case of chest tomosynthesis typically 40-60 projections are acquired, the angular range of projections is limited to ±20°, and the total thickness (extension along the *z* axis) of the examined volume is not less than 250 mm. If only a few (e.g. 60), evenly spaced slices of the volume are reconstructed there will remain parts of the volume that have not been reconstructed in any slice, mainly in higher frequencies. Fig. 4 illustrates the main lobe of the aggregated MTF of two adjacent slices in the case of 60 (in the upper picture) and in the case of 400 (in the lower picture) reconstructed slices. The aggregated MTF is:

$$\mathbf{M}'^{(i)}=\max\left\{\mathbf{M}^{(i)},\mathbf{M}^{(i+1)}\right\} \tag{15}$$

where $\max\{\cdot,\cdot\}$ denotes the elementwise maximum operator.

From the figure it is clearly visible that if only 60 slices are

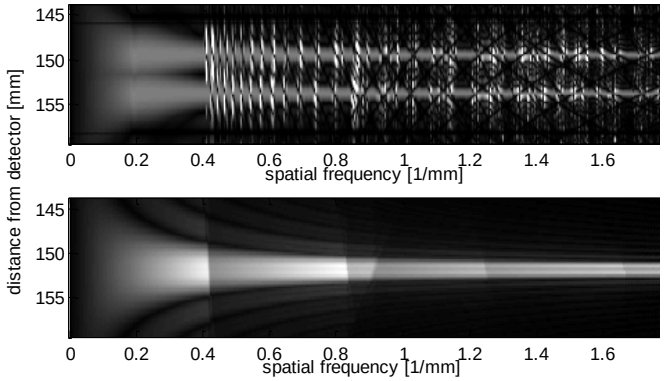


Fig. 4. Central lobe of a typical aggregated MTF of adjacent reconstructed coronal slices. In the case of the upper picture 60 slices are reconstructed, while in the case of the lower 400 slices are reconstructed from the same projections.

mm from the detector are not reconstructed into any of their two enclosing slices at >0.4 1/mm frequencies. In the second case (400 slices are reconstructed) such a problem did not occurred. Fig. 5. shows reconstructions of an ideal spherical artificial chest nodule (with the diameter of 3.5 mm) located at an anthropomorphic phantom. The visibility (therefore the detectability) of the nodule is significantly lower in the case of 60 slices compared to the case of 400 slices. This phenomenon is not the same, but similar to the so called partial volume artifact [7], therefore we use the same term.

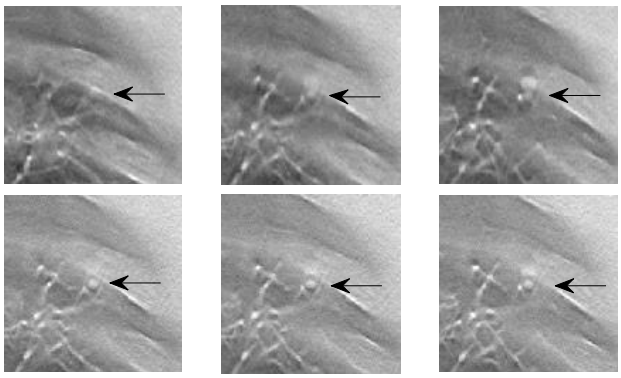


Fig. 5. Adjacent 3 slices of the same reconstructed volume. In the upper row 60 slices, in the lower row 400 slices were reconstructed. The arrows point to the nodule's reconstruction.

V. Results and Discussions

The accumulated extent of the parts of the examined volume which is not reconstructed by any of the slices as in-plane signal (not covered by the union of the main lobes of the reconstructed slices' MTF) at a given spatial frequency is:

$$V_{(\omega)} = \left(H - \sum_i \int f_{(\omega,i)}(l) \cdot g_{(i)}(l)\, dl \right) \Big/ (2H) \qquad (16)$$

where H denotes the thickness (extent along the *z* axis) of the examined volume, $g_{(i)}(l) = ind\left(h(i-1) \le l \le h(i+1)\right)$ and $f_{(\omega,i)}(l)$ is defined by the full width half maximum principle:

$$f_{(\omega,i)}(l) = ind\left(\mathbf{M}^{(i)}_{(\omega,p(l))} \ge \max_{p(h(i-1)) \le v \le p(h(i+1))} \left\{ \mathbf{M}^{(i)}_{(\omega,v)} \right\} \Big/ 2 \right) \qquad (17)$$

$ind(\cdot)$ denotes the logical indicator function. Its value is 1 if and only if its argument is true, else it is zero. Based on our observations $V_{(\omega)}$ significantly depends on the number of reconstructed slices. v as the function of spatial frequency (denoted by ω) and as the function of the number of the reconstructed slices is illustrated by Fig. 6.

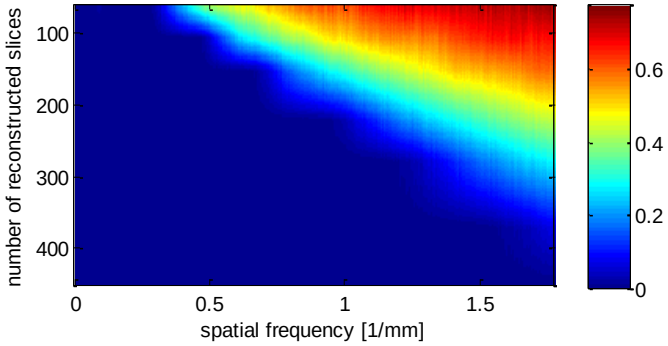


Fig. 6. The relative extent of the volume, which is not reconstructed as in-plane signal by any of the slices as a function of the spatial frequency and the number of reconstructed slices.

From the figure we can conclude that to compute approx. 350-400 slices is necessary in order to completely avoid the problem of partial volume in the case of chest tomosynthesis reconstructed by the MITS algorithm. The recommended minimum number of the slices depends on the thickness of the examined volume and the thickness of the reconstructed slices. This concrete result corresponds to 250 mm thick chests reconstructed by the previously described MITS algorithm. For linear shift invariant reconstruction methods the optimal parameters of the reconstruction in the sense of minimizing the partial volume artifact can be determined by the previously described MTF based method.

References

[1] D.G. Gant, "Tomosynthesis: a three-dimensional radiographic imaging technique," in IEEE Trans. Biomed. Eng., 19, 20-28, 1972.

[2] D.J. Godfrey et al, "Practical strategies for the clinical implementation of matrix inversion tomosynthesis," in Proc. SPIE Physics of Medical Imaging 5030, 379-39, 2003.

[3] B. G. Ziedes des Planets, "Eine neue methode zur differenzierung in der roentgenogreaphie (planigraphie)," in Acta Radiologica, 13, 182-192, 1932.

[4] D.J. Godfrey et al, "Stochastic noise characteristics in matrix inversion tomosynthesis (MITS)," in Med. Phys., 36 (5), 1521-1532 2009.

[5] D.J. Godfey et al, "Optimization of the matrix inversion tomosynthseis (MITS) impulse response and modulation transfer function characteristics for chest imaging," Med. Phys., 33, 655-667, 2006.

[6] D. Hadházi et al, "Measurement Quality of MITS reconstructed Tomosynthesis Slices Depending from Parameters of Images Acquiring and Reconstruction Algorithm," in IFMBE Proc. 45, 114-117, 2015.

[7] M. Soret et al, "Partial-Volume Effect in PET Tumor Imaging," in Journal of Nuclear Medicine, 48 (6), 932-945, 2007.

On-Line Detector Connectivity Check Based on a Parasitic Signal Coupling

Csaba F. Hajdu*†, Tamás Dabóczi†, Christos Zamantzas*
*European Organization for Nuclear Research (CERN), Geneva, Switzerland. Email: {cshajdu,czam}@cern.ch
†Budapest University of Technology and Economics, Department of Measurement and Information Systems, Budapest, Hungary. Email: {chajdu,daboczi}@mit.bme.hu

***Abstract*—Beam loss monitoring (BLM) is a key element of the scheme for machine protection and beam setup at the European Organisation for Nuclear Research (CERN). The project related to this paper aims to elaborate a process ensuring a comprehensive and continuous surveillance of the entire BLM signal chain, particularly the functionality and connectivity of the detectors. This paper presents a method elaborated with the view to realizing a noninvasive detector connectivity check.**

I. Introduction

The European Organisation for Nuclear Research (CERN) is one of the world's leading particle physics research laboratories, hosting a complex of particle accelerators dedicated to fundamental research. The particles injected into the flagship machine of the complex, the Large Hadron Collider (LHC), are first accelerated to progressively increasing energies through a series of preaccelerators known as the LHC injectors.

These aging low energy machines are being overhauled and consolidated within the framework of the LHC Injectors Upgrade (LIU) project in order to meet the more and more stringent requirements that the evolution of the LHC imposes on the quality of the particle beams.

The measurement of the showers of secondary particles generated by particles under acceleration escaping the beam lines is referred to as beam loss monitoring (BLM). The strategy for machine protection and beam setup at CERN relies heavily on BLM systems. A continuous supervision of the entire BLM signal and processing chain is therefore fundamental, yet no particle accelerator in the world has this feature at present.

This paper presents the latest results of a project aimed at designing a process providing such continuous surveillance for the new BLM system currently under development for the injectors, mandated by the LIU project.

II. Beam loss monitors

A. Beam loss monitoring at CERN in brief

The beam loss monitoring system of the LHC uses mainly ionization chambers as detectors. The ionizing particles crossing the gas-filled active volume of the chamber create electrons and ions, which are separated by a bias high voltage applied to the extremities of the detector and collected on a stack of parallel electrodes. The resulting current signal is acquired and digitized by the front-end cards of the system, then forwarded to the back-end electronics for further processing. The measured data are archived for machine tuning and calibration purposes, and if necessary for protecting the machines from damage caused by the beam, the safe extraction of circulating beams and an inhibition of further injections is initiated.

The new BLM system for the injectors is in an advanced stage of development at present. Currently, two prototypes of the system are installed: one in the laboratory, and one at the Proton Synchrotron Booster (PSB) accelerator alongside the operational system currently used for machine protection. Each front-end card of the BLM system for the injectors is capable of acquiring the output current of eight detectors [1]. The digitized signals are forwarded to the back-end card over a bidirectional optical link for further processing. These cards feature reprogrammable FPGA devices to ensure flexibility and high data throughput. The acquisition stage of the new system has been designed to be flexible and accommodate various detector types. In most locations, ionization chambers similar to those in the LHC system will be installed [2].

B. Connectivity checks

The current best solution for supervising the functioning of a beam loss monitoring system along with the correct connection and functionality of its detectors is in operation at the LHC, where a connectivity check of each detector channel is enforced every 24 hours. This procedure, only executable while the accelerator is offline, involves inducing a sinusoidal modulation on the order of $10\,\mathrm{mHz}$ in the bias high voltage of the ionization chambers, then measuring the resulting modulation in the output current of each channel. If the cabling connection of a detector is defective, the harmonic modulation will not be detectable in its output current. Additionally, correspondence has been found between variations in the amplitude and phase of the modulation in the output signal and various nonconformities of the acquisition chain, thus, this process is also usable as a component integrity survey.

The project related to this paper targets the elaboration of an improved procedure, to be constructed, tested and integrated into the BLM system of the injectors. The process should provide a continuous supervision of the entire signal chain from the detectors to the processing electronics as well as the measurement ability of the detectors.

Tamás Dabóczi acknowledges the support of ARTEMIS JU and the Hungarian National Research, Development and Innovation Fund in the frame of the R5-COP project.

III. The suggested measurement method

A. Approach for a non-invasive connectivity check

A series of measurements have been conducted on the prototype of the BLM system for the injectors to assess the capabilities of the signal processing chain [3], with a view to implementing a connectivity check process relying on modulating the high voltage supply of the detectors, similarly to the LHC implementation. This investigation has revealed that parasitic components corresponding to the switching frequency of about 30 kHz of the high voltage power supply (HVPS) and its harmonics are consistently present in the digitized output signal of the detectors. The detection of these parasitic components is very appealing, since it may allow verifying detector connectivity continuously by simply post-processing the digital signal without altering the analog signal in any way, thereby realizing a non-invasive on-line connectivity check. Further studies will be required for assessing how these components are coupled into the signal chain and which connection nonconformities can be detected by this approach. The part of the work described in the present paper is aimed at detecting this parasitic signal efficiently within the hardware constraints imposed by the architecture of the BLM system.

The resonator-based spectral observer (Fourier analyzer, FA) proposed by Péceli [4] was chosen as basis for a signal processing method to detect the spectral peaks resulting from the operation of the HVPS. Its advantages include inherent stability due to the unit negative feedback in the observer, flexibility in setting the frequencies of the observer channels [5], and that a straightforward frequency adaptation method [6] has been suggested for it, which should allow tracking the slowly varying frequency of the HVPS components.

B. The adaptive Fourier analyzer

One obtains the Fourier analyzer when using Péceli's resonator-based observer as a spectral observer. The structure estimates the state variables of the conceptual signal model [4], which can in turn be interpreted as a multisine generator reconstructing the signal from its DFT. In this interpretation, the state variables of the observer correspond to the complex Fourier coefficients of the signal and they each represent a harmonic resonator of the corresponding frequency.

The state variables, i.e. the complex Fourier components of the signal y_n can then be estimated using an appropriately designed observer, of the structure shown in Fig. 1, described by the following system equations:

$$\hat{\mathbf{x}}_{n+1} = \hat{\mathbf{x}}_n + \mathbf{g}e_n = \hat{\mathbf{x}}_n + \mathbf{g}\left(y_n - \hat{y}_n\right), \tag{1}$$

$$\hat{\mathbf{x}}_n = \left[\hat{x}_{i,n}\right]^T \in \mathbb{C}^{N\times 1}, \qquad i = -K, \ldots, K, \tag{2}$$

$$\hat{y}_n = \mathbf{c}_n^T \hat{\mathbf{x}}_n, \tag{3}$$

$$z_i = e^{j2\pi i f_0}, \tag{4}$$

$$\mathbf{c}_n = \left[c_{i,n}\right]^T \in \mathbb{C}^{N\times 1}, \qquad c_{i,n} = e^{j2\pi i f_0 n} = z_i^n, \tag{5}$$

$$\mathbf{g}_n = \left[g_{i,n}\right]^T \in \mathbb{C}^{N\times 1}, \qquad g_{i,n} = \frac{\alpha}{N} c_{i,n}^*, \tag{6}$$

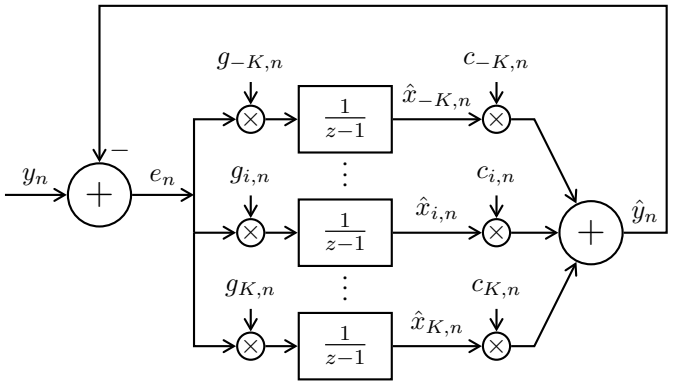


Figure 1. Block diagram of the resonator-based observer.

where $\hat{x}_{i,n}$ are the estimators of the state variables, thus of the complex Fourier components of the signal, $\hat{y}_n$ is the estimated signal and e_n is the estimation error. The vector $\mathbf{c}_n$ is the time-varying coupling vector to calculate the estimator of the signal from the state variables, while $\mathbf{g}_n$ represents the observer gain, a tunable parameter for setting the poles of the observer. The signal is considered to have K harmonics, thus $N = 2K + 1$ complex Fourier components including DC. For a real valued signal, the estimators of these components will form complex conjugate pairs: $\hat{x}_{i,n} = \hat{x}^*_{-i,n}$. The frequency of the fundamental harmonic relative to the sampling frequency is $f_0 = f_1/f_s$. It is assumed that $K \cdot f_1 \leq f_s/2$. If they're equal, $i = -K, \ldots, K+1$ and $N = 2K+2$ in all equations.

At this point, suppose that the frequency of the fundamental signal harmonic f_0 and the frequency of the fundamental resonator f_r, both expressed relative to the sampling frequency, are different. Then, the complex number estimating the Fourier coefficient of the fundamental harmonic is going to rotate with a frequency corresponding to the difference between f_0 and f_r [6], that is,

$$f_0 - f_r \approx \frac{\arg \hat{x}_{1,n+1} - \arg \hat{x}_{1,n}}{2\pi}, \tag{7}$$

which constitutes the basic principle of the methods for adapting f_r.

C. Realizing a customized spectral observer

As mentioned in Section II-A, front-end and back-end processing of the digitized data in the BLM system of the injectors is based on reconfigurable FPGA devices. This implies that ideally, data processing for on-line detector connectivity checks is realized on these FPGAs.

It's clear from (1)–(6) that the state update of the observer requires multiplications and additions of complex numbers. The polar representation ($z = Ae^{\mathrm{j}\varphi}$) would be convenient for multiplications, however, additions would require converting the numbers to the algebraic form ($z = a + \mathrm{j}b$), which is extremely resource-consuming on an FPGA. In the algebraic form, complex multiplications can be realized by a series of multiplications and additions, thus using this representation is preferable in terms of FPGA resource cost.

The digitized current values are represented using 20-bit integer arithmetic. For the FPGA implementation, we decided to use IEEE-754 compliant 32-bit floating point arithmetic realized with manufacturer-supplied IP cores to cope with the dynamic range expected from the inputs and the coefficients used for the state update.

For implementing the observer, we didn't rely on the linear time-variant (LTV) model presented in Section III-B. The observer model we chose is linear time-invariant (LTI) since it uses time-invariant $\mathbf{g}_n$ and $\mathbf{c}_n$ coefficients, but it's equivalent to the LTV model. It has the advantage that exponentiation of z_i as in (5) is not required. The problem with this calculation is that it is numerically unstable when using the algebraic form with 32-bit IEEE-754 arithmetic for certain values of $c_{i,1}$: $\lim_{n\to\infty} c_{i,n} = 0$ or ∞.

Our current implementation includes two parallel processing channels calculating the real and imaginary parts of the updated state variable separately. With a multiplier, an accumulator and a memory block storing the LTI coefficients, using the pipelining features of the floating point cores, each channel is capable of producing a new value every three clock cycles once the pipeline is full. The updated values for each state variable are calculated sequentially. This processing could be carried out in parallel for each resonator, and if necessary, all operations can be performed using just a single channel, which affords a big margin for tuning FPGA resource use versus processing speed.

The original observer structure was tailored heavily for an FPGA-based implementation. Since the output current of the detectors is acquired at a frequency of $500\,\mathrm{kHz}$ in the BLM system of the injectors, and the aim is to acquire a signal of about $30\,\mathrm{kHz}$ and its harmonics, the sampling frequency cannot be reduced by much. The observer runs at $500\,\mathrm{kHz}$ to ensure on-line data processing. Using sensible DFT bin widths of about $10 - 100\,\mathrm{Hz}$, this would yield an unmanageable number of resonators, since either the FPGA resource usage or the time required for processing would be too high. Thus, we considered an extremely coarse resonator distribution with the DFT points placed at the HVPS frequency and its harmonics. If slower convergence can be tolerated, the bandwidth of the resonator channels may be controlled by tuning the $\frac{\alpha}{N}$ parameter. This allows achieving an acceptable frequency sensitivity as shown in Fig. 2, without impacting on the stability of the structure.

In order to have an estimate of the noise floor in the measurement along with the value of the signal component of interest, we turned some of the individual resonators at lower frequencies into resonator triplets by adding two resonator positions on either side of the resonator in question. However, if the observer gain $\frac{\alpha}{N}$ is set so high that the bandwidths of neighboring resonators overlap, resonances may occur.

D. The new method for frequency adaptation

1) Principle: Using (7) directly for frequency adaptation would require calculating the phases of complex numbers, which is a resource-consuming task better avoided. Moreover,

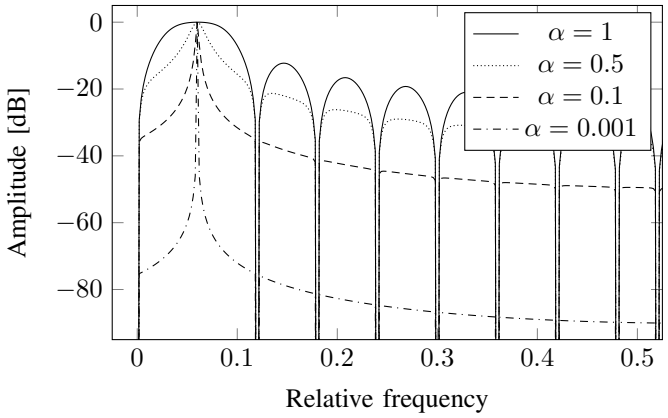


Figure 2. Transfer function of a resonator channel for different values of α. Its center frequency is $f_{rel} = 0.06$, that is, $30\,\mathrm{kHz}$ at $f_s = 500\,\mathrm{kHz}$. The transfer function has zeroes at the frequencies of the other resonators.

the phase of the Fourier coefficients tends to have abrupt transients when processing the output current of the detectors, which might misguide the adaptation. Therefore, we devised a method to adapt the frequency without having to calculate actual phases: we detect zero crossings in phase. This is very simple when using the algebraic form, since one only needs to consider the sign bits of the real and imaginary parts of the complex number. We need to make sure we exclude those events when the phase wraps from π to $-\pi$ or vice versa, since technically, the phase crosses zero even in these cases. Taking this consideration into account, we register an event if $\mathrm{Im}\{\mathrm{x}_{1,\mathrm{n}}\}$ changes sign while $\mathrm{Re}\{\mathrm{x}_{1,\mathrm{n}}\} \geq 0$. Then, if N_z zero crossings have been registered in a time window of t_m, we have the following estimate for the frequency difference:

$$|f_0 - f_r| \approx \frac{(N_z - 1)\cdot 2\pi}{t_m \cdot f_s}\,\frac{1}{2\pi} = \frac{N_z - 1}{t_m \cdot f_s}, \qquad (8)$$

where the sign of the difference can be determined by considering the *sign* of the zero crossing: if $\mathrm{Im}\{\mathrm{x}_{1,\mathrm{n}}\}$ changes from negative to positive, we register a positive zero crossing corresponding to $f_0 > f_r$, and the opposite for negative zero crossings. At the expense of slower adaptation, this scheme can be implemented at a comparatively low cost in terms of FPGA resources and it's also less subject to the effect of abrupt transients in the phase signal.

2) Implementation: The source of the parasitic signal we're aiming to detect is the same HVPS for all eight channels of the same card, thus we decided to carry out frequency adaptation on one channel only and use the same coefficient values for all channels.

The frequency adaptation principle described in Section III-B relies on the LTV model of the observer. Our realization calculates LTI state variables, which thus need to be multiplied by z_i^n for conversion to LTV. This exponentiation is subject to the numerical problems described in Section III-C. Since z_i is recalculated every time the frequency is adapted, this effect is mitigated but not eliminated.

If the input signal is noisy, repetitive fake zero crossing events like those shown in Fig. 3 may occur. These can be eliminated by setting an adequate minimum time difference requirement between consecutive zero crossings and ignoring those violating this constraint.

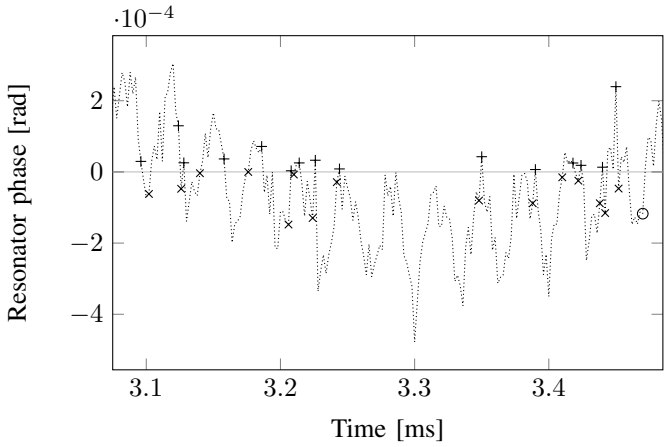


Figure 3. Spurious zero crossings caused by noise in the phase of $\hat{x}_{1,n}$ calculated from a real signal acquisition. The markers × and + correspond to rejected positive and negative zero crossings, respectively, while ○ represents the one accepted positive zero crossing.

Additionally, transients in phase resulting from noise may cause the processing to register alternating positive and negative zero crossings. In our scheme, we count these events separately, either until a given number of events of either type is collected or for a given amount of time, and then we use the difference $N_{z+} - N_{z-}$ to calculate the estimate of the frequency difference. Currently, we acquire 128 events of either type, adapt the frequency, then discard the first 128 events to suppress the transients caused by the adaptation.

Since calculating the new set of coefficients using FPGA logic directly would imply prohibitively high resource usage, we decided to carry out this processing on a very simple soft-core CPU embedded in the FPGA. Since the CPU features no floating-point core, all floating-point arithmetic is executed in software, therefore calculating a new set of coefficients for the 13 resonators we currently use takes about $0.1\,\mathrm{s}$. Since the adaptation itself is fairly slow, this time overhead is acceptable.

E. Results with the custom adaptive observer

We tested the current version of our processing with frequency adaptation both in the laboratory system and the prototype installed at the PSB.

The performance of the frequency adaptation in the laboratory system is demonstrated in Fig. 4. The processing is capable of following the slow frequency drift satisfactorily.

As a tradeoff between limiting the noise in the signal and ensuring the detectability of the HVPS contributions, our resonators have a bandwidth of about $5-10\,\mathrm{Hz}$. The resonator outputs react to beam loss events, but the recovery is fairly quick (about $0.1-0.5\,\mathrm{s}$) and these contributions can be filtered out. They also have little effect on frequency adaptation. In general, the resonators centered on the HVPS peak show values about $20\,\mathrm{dB}$ higher than the other resonators of the triplet if the HVPS contributions are present.

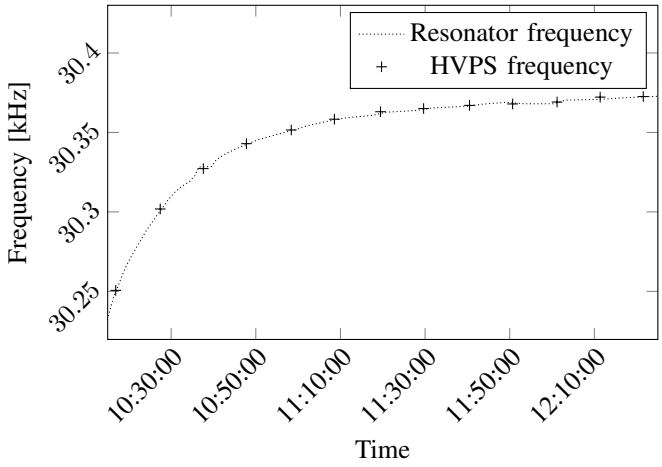


Figure 4. Demonstration of frequency tracking by the resonator-based observer: evolution of the fundamental resonator frequency, registered at each frequency adaptation step. The variation of the fundamental frequency of the HVPS contributions is also shown: every marker (+) represents an acquisition.

Under real operating conditions, our method usually seems to underestimate the actual frequency difference, especially if it is relatively high. However, we've seen the frequency adaptation converge for initial differences as high as $500\,\mathrm{Hz}$. In our experience, this covers the frequency band the HVPS signal fluctuates in.

IV. Conclusions

We presented a method to identify parasitic components caused by the operation of the high voltage power supply in the output current of the BLM detectors. Further studies will need to be conducted to determine precisely how this signal is coupled into the signal chain and which connection nonconformities can be detected by this approach. An extension of the procedure to cover all faults targeted by the LHC implementation is foreseen.

References

[1] W. Viganò, M. Alsdorf, B. Dehning, M. Kwiatkowski, G. G. Venturini, and C. Zamantzas, "10 Orders of Magnitude Current Measurement Digitisers for the CERN Beam Loss Systems," *J. Instrum.*, vol. 9, p. C02011. 10 p, Sep 2013.

[2] C. Zamantzas, M. Alsdorf, B. Dehning, S. Jackson, M. Kwiatkowski, and W. Viganò, "System Architecture for measuring and monitoring Beam Losses in the Injector Complex at CERN," Tech. Rep. CERN-ACC-2013-0252, CERN, Geneva, Aug 2012.

[3] C. F. Hajdu, T. Dabóczi, and C. Zamantzas, "Design and implementation of a real-time system survey for beam loss monitoring systems," in *XXI IMEKO World Congress - Full Papers*, pp. 1176–1180, 2015.

[4] G. Péceli, "A common structure for recursive discrete transforms," *Circuits and Systems, IEEE Transactions on*, vol. 33, pp. 1035–1036, Oct 1986.

[5] G. Péceli, "Sensitivity properties of resonator-based digital filters," *Circuits and Systems, IEEE Transactions on*, vol. 35, no. 9, pp. 1195–1197, 1988.

[6] F. Nagy, "Measurement of signal parameters using nonlinear observers," *Instrumentation and Measurement, IEEE Transactions on*, vol. 41, pp. 152–155, Feb 1992.

Generating Unit Isolation Environment Using Symbolic Execution

Dávid Honfi, Zoltán Micskei
Budapest University of Technology and Economics
Department of Measurement and Information Systems
Budapest, Hungary
Email: {honfi, micskeiz}@mit.bme.hu

***Abstract*—Research of source code-based test input generation has recently reached a phase, where it can be transferred to industrial practice. Symbolic execution is being one of the state-of-the-art techniques, yet its usage on industrial-sized software is often hindered by several factors, like the interaction with the environment of software under test. The solution of this problem can be supported with isolating the interactions by using so-called test doubles instead of the original objects. This time-consuming process requires deep knowledge of the unit under test. The technique presented in this paper is able to automatically generate isolation environment from the data collected during symbolic execution. We also present our promising preliminary results using a prototype implementation.**

I. Introduction

Generating test inputs and test cases from source code has been one of the main topics in software test research since decades. Numerous techniques and algorithms have been proposed to enhance the test generation processes by analyzing only the source code itself, commonly called as white-box test generation. Symbolic execution [1] is one of the state-of-the-art techniques due to the promising results of its fault detection ability. This technique represents possible paths of the source code with logical formulas over symbolic variables. The execution starts from an arbitrary method in the program and each statement is interpreted in parallel with gathering the expressions over the symbolic variables. The solution of these collected expressions provide input values that drive the execution of a program along different paths. Test cases are formed using these inputs extended with assertions derived from the observed behavior of the program under test.

In our previous research, we used Microsoft Pex [2] (currently known as IntelliTest) for test generation that is a state-of-the-art white-box test generator tool for .NET. Pex uses dynamic symbolic execution, which is an enhanced technique that combines concrete with symbolic execution. The tool generates input values for so-called Parameterized Unit Tests (PUTs) [3] that are simple unit test methods with arbitrary parameters. PUTs can be extended with assumptions and assertions, thus can serve as a test specification. Pex generates test cases from the combination of the generated inputs and the corresponding PUTs.

A current research topic is the industrial adoption of symbolic execution. However, it is hindered by several factors [4]–[7]. One of the main problems is that in the majority of cases, the test cases generated by symbolic execution typically achieve very low source code coverage. Our previous experiences [8] also confirmed these challenges. We applied Pex in testing of a model checker tool and a content management system.

A solution for this problem could be isolating the external dependencies of the unit under test (see Section II). However, in large-scale software that contains numerous components interacting with each other, a plentiful of calls to the environment can be identified. The identification and isolation of these calls is a highly time-consuming task, especially for test engineers who did not participate in the development of the unit.

In this paper, we present an approach and a prototype tool that endeavors to support symbolic execution-based test generation in complex, environment-dependent software by automatically generating code for isolation purposes. Thus, the research question of the paper is the following.

How can the isolation process be supported during symbolic execution-based test generation?

II. Background

Unit-level testing should be done in isolation, thus all the external dependencies of the unit should be removed or replaced. A solution could be to replace the external dependency with a *replacement object*, and call into that instead of the original. This idea and the increasing importance of unit testing led to a whole new area in software test engineering called *test doubles*.

Test doubles is the common name of static or dynamic objects that can be used as a replacement of real objects during test executions, in order to handle the problem of isolation in unit testing. Many types of test doubles exist, however the naming conventions can be different across publications. To overcome this, Meszaros wrote a summarizing book [9], that assesses the notions and patterns around unit testing, including test doubles too. We also applied the notions of this book extended with a Microsoft-specific type, called shims. Shims are powerful test doubles where the call is made to the original object, however the call is detoured during runtime to a user-defined method. This allows easy isolation of calls that invoke 3rd party libraries, legacy code or other resources where source is not available.

Consider the following example scenario introducing the isolation problem. Let `UUT` be the unit under test, and let `ED` be the external dependency. The body of method `M1(int):int` contains a branching where the decision depends on the return value of method `M2(int):int` from `ED`. Unit testing class `UUT` shall include the isolation of the mentioned dependency, moreover this shall be applied during symbolic execution-based test generation too. If method `M2` is not accessible during testing due to some reason (e.g., not yet implemented, accesses external resources) symbolic execution would not reach the statements found in the two branches.

```
class UUT
{
    ED ed = new ED();
    int M1(int a)
    {
        if(ED.M2(a)) return 1;
        else return -1;
    }
}
```

If we assume that parameter `a` will not be larger than 10 and only -1 or 1 can be returned, then creating a PUT for method `M1` would look like the following.

```
void M1Test(int a)
{
    Assume.IsTrue(a <= 10);
    UUT uut = new UUT();
    int result = uut.M1(a);
    Assert.IsTrue(result == 1 || result == -1);
}
```

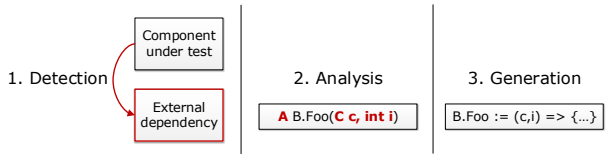
PUTs are used for compact representation of multiple test cases, where different input values trigger different behavior of the unit under test. It can be seen however that the PUT misses the isolation in this example. For the solution, we use the isolation syntax defined by Microsoft Fakes, a powerful isolation framework for .NET that uses shims, thus can isolate wide-range of external invocations. Extending the PUT with the following snippet will isolate the external call of `M2` for every instance of `ED` and return 1 or -1 depending on the value of the passed parameter, thus simulating a custom behavior.

```
ED.AllInstances.M2 = (ED instance, int param) =>
{ return param > 5 ? 1 : -1; };
```

III. Overview of the Approach

In order to support symbolic execution-based test generation, our approach is to generate the source code of the isolation environment automatically. This novel technique uses the collected data from the symbolic execution process itself.

Fig. 1. The approach of automated isolation



The technique builds on top of parameterized unit tests in order to have test doubles, which can give back values that are relevant to the component under test. A quick overview of the approach is presented in Figure 1.

The automated generation of isolation environment relies on an analysis process, which is conducted when an invocation to a predefined external dependency is reached during the symbolic execution. Then, based on the results of the analysis, the generation step creates double objects that are able to replace the original ones.

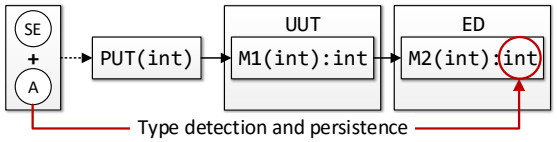
A. Detection

Detection is the first phase of the isolation process. Firstly, the test engineer defines the unit or namespace under test with giving its fully qualified name (FQN). During the symbolic execution, this FQN is used for detecting an external call by analyzing the reached stack frames. When an external invocation is detected, all the information regarding this call is collected and stored for later usage by the analysis step.

B. Analysis

The analysis step plays the main role, since the decisions are made here that define how double objects are generated. The analysis can be divided into three substeps, which are the *analysis of return value*, the *analysis of parameters* and the *assessment* of the results gathered from the previous steps.

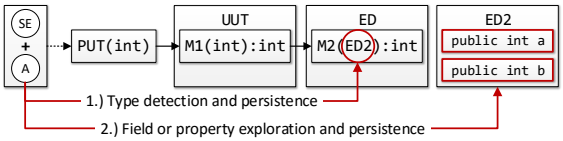
Fig. 2. Overview of return value analysis example



1) Return value analysis: In the first step, the return value of the invoked external method is analyzed, which is a crucial information of the double object. Figure 2 shows the overview of the code example mentioned in Section II, where SE denotes the symbolic execution process and A stands for our analysis technique. The return value can be used in the unit under test in logical conditions, thus execution paths possibly rely on this value. In order to cover these paths, the correct value must be selected. If a path relies on the variable that obtained its value from the external call, the symbolic execution interprets it as a term in the path condition. Problems occur, when the analysis can not provide proper inputs through this dependency (e.g., not yet implemented, gets value from database or file system). This can be alleviated if the solver of symbolic execution can give concrete values for the variable that represents the return value. By this way, arbitrary values can be assigned to this variable and the coverage criteria (e.g., an execution path) can be satisfied that depends on the variable. The arbitrary values can be passed to the concrete execution through the parameters of the unit test. In summary, our idea provides a solution with the analysis of the return value, where two actions are done.

- The parameters of the PUT is extended with the return type of the external dependency.
- A test double is created in order to replace the behavior of the original class. In the body of the double, the new PUT parameter is returned, which gives the ability for symbolic execution to handle it as a free variable that can have arbitrary values.

Fig. 3. Overview of parameter analysis example



2) Parameter analysis: The analysis of parameters is the second part of the analysis, however not all types of parameters are in focus. Method parameters can be primitive or complex types. In the two popular managed environments (.NET, Java) every complex type is handled as reference and the parameters are passed by value by default. Thus, when using reference type parameters, the reference itself is passed to the method as value, which means it is copied and refers to the same object. This enables the called method to modify the pointed object, which modifications can be also seen in the caller. However, the original reference cannot be modified. The reference type parameters lead to a problem in isolation scenarios, when the called method is an external dependency, because the passed object can be modified inside the dependency and therefore it can affect the coverage in the unit under test. Our idea to overcome this is similar as in the case of return values, but the scenario is more complex. The first step is the same: extending the parameters of the PUT with the complex type parameter under analysis and handle it in the created double object. However, due to the complex type, there are numerous possibilities to modify the state of the object outside the unit. The idea is to explore the publicly available fields and properties of the object and use them to alter its state. By this way, the generated results of our approach can simulate the actions made inside the external dependency that can be required to increase the coverage inside the unit under test. Figure 3 presents this process on the extension of the example found in Section II. The scenario contains a new `ED2` external dependency as a complex type parameter.

3) Assessment: During the last step of the analysis, all the collected information about the return values and parameters are filtered for duplications, then stored, which is used for double generation. Every method should contain the information that describe what to emit, when they are in the focus of code generation, which also includes the doubles of complex type parameters.

C. Generation

The generation is the last step of processing an external dependency, which can be also divided into substeps. Firstly, the newly created parameters of the parameterized unit tests are emitted and appended to the original one. Then, the double of the method is assembled and emitted into the body of the PUT. This emission includes the name of the double method, which can be specific to isolation frameworks and also includes the inner body that can include setting of state modification for parameters and verification too. Finally, the test doubles of the complex type parameters are generated that are property or field setter methods (if the type of the parameter is located outside the unit under test).

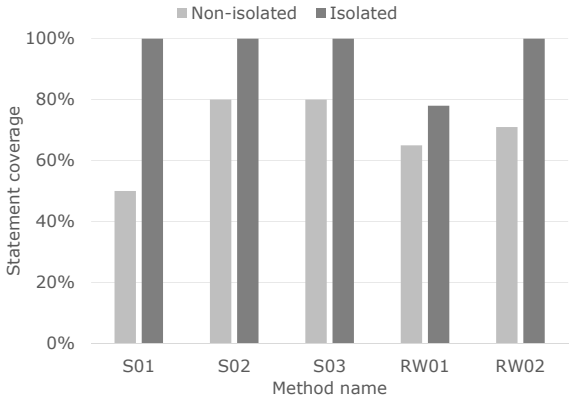
IV. Preliminary Evaluation

We implemented the presented technique as an extension to Microsoft Pex and Fakes. Fakes seamlessly collaborates with Pex and is powerful enough to use for our approach. Fakes is capable of creating stubs and shims for a very wide-range of method calls (regardless their place and type) found in any .NET software.

The implementation had many challenges including the run-time reflection of types and methods. We used this technique to obtain detailed information on each external call that is required for the return value and the parameter analysis.

Although the current prototype implementation does not support every scenario that can be found in arbitrary .NET code, it does work sufficiently for preliminary evaluation. We implemented example scenarios (intentionally similar to real-world source code snippets) to present the potential in our approach. The coverage results with and without our technique is presented in Fig 4. This figure shows the achieved statement coverage with and without the generated isolation environment after symbolic execution on the five different method under test. The names starting with *S* denotes simple scenarios where symbolic execution can benefit from the isolation. The two methods denoted with *RW* represent source code gathered from open-source software ([10] and the CMS system mentioned in Section I. There were no hindering factors during the measurement, however we wanted to make sure that our results are valid: the evaluation was repeated three times on each method. The presented figure shows the average statement coverage of the three executions.

Fig. 4. The results of the preliminary evaluation



The results show that in each case, running Pex with the generated isolation environment achieves higher statement coverage than running the tool without any unit isolation. This may emphasize that this technique has potential in the area of supporting the usage of symbolic execution in large-scale industrial software.

V. Related work

Our idea originally derives from a paper written by Tillmann *et al.* [11], where the idea of mock object generation is described. They also created a case study for file-system dependent software [12], which showed promising results. Their technique is able to automatically create mock objects with behavior and ability to return symbolic variables, which is used during the symbolic execution to increase the coverage of the unit under test. However, their solution needs the external interfaces explicitly added to the parameterized unit tests, moreover they did not consider reference type parameters that can affect the coverage. Thus, our solution covers a wider area of scenarios and needs rather less user interaction for the automated generation (our approach only requires the namespace of the unit under test).

The idea of Galler *et al.* is to generate mock objects from predefined design by contract specifications [13]. These contracts describe preconditions of a method, thus the derived mocks are in respect of them, which makes mocks able to avoid false behavior. However, their approach does not relate to symbolic execution, and it may also introduce work overhead to create contracts. A similar approach is introduced in parallel with a symbolic execution engine to Java by Islam *et al.* [14]. The difference is that they build on interfaces as specifications instead of contracts.

An other approach of mock generation was presented by Pasternak *et al.* [15]. They created a tool called GenUTest, which is able to generate unit tests and so-called mock aspects from previously monitored concrete executions. However, the effectiveness of the approach largely relies on the completeness of previous concrete executions, while our approach relies on symbolic execution.

VI. Conclusion and future work

One of the discovered challenges in real-world scenarios was the unit isolation in testing a software component, because these applications have several external dependencies (e.g., databases, external services). Isolating the dependencies requires large amount of time, which can be reduced by automation.

The described isolation technique in this paper could support the solution of this problem by automatically generating isolation environment. The main idea is to detect the dependencies during the symbolic execution and to generate the isolation environment for the unit under test from the data retrieved from symbolic execution. We also presented our preliminary evaluation of a prototype implementation that showed promising results.

The approach presented in this paper may be continued in the following directions.

- Expanding the implementation to cover all the possible unit isolation cases that could be present in real-world software.
- Experiments and measurements for the presented technique to confirm its usability in different scenarios.
- Combination of automated isolation and compositional symbolic execution [16] that leads to a new level of automated test input generation, where the work of test engineers can be greatly supported.

References

[1] J. C. King, "Symbolic Execution and Program Testing," *Commun. ACM*, vol. 19, no. 7, pp. 385–394, 1976.

[2] N. Tillmann and J. de Halleux, "Pex–White Box Test Generation for .NET," in *TAP*, ser. LNCS, B. Beckert and R. Hähnle, Eds. Springer, 2008, vol. 4966, pp. 134–153.

[3] J. de Halleux and N. Tillmann, "Parameterized Unit Testing with Pex," in *TAP*, ser. LNCS, B. Beckert and R. Hähnle, Eds. Springer, 2008, vol. 4966, pp. 171–181.

[4] X. Qu and B. Robinson, "A Case Study of Concolic Testing Tools and their Limitations," in *Empirical Software Engineering and Measurement (ESEM), 2011 Int. Symposium on*, Sept 2011, pp. 117–126.

[5] G. Fraser, M. Staats, P. McMinn, A. Arcuri, and F. Padberg, "Does automated white-box test generation really help software testers?" in *Proc. of the 2013 Int. Symposium on Software Testing and Analysis*, ser. ISSTA 2013. ACM, 2013, pp. 291–301.

[6] P. Braione, G. Denaro, A. Mattavelli, M. Vivanti, and A. Muhammad, "Software testing with code-based test generators: data and lessons learned from a case study with an industrial software component," *Software Quality Journal*, vol. 22, no. 2, pp. 1–23, 2013.

[7] N. Tillmann, J. de Halleux, and T. Xie, "Transferring an Automated Test Generation Tool to Practice: From Pex to Fakes and Code Digger," in *Proc. of the 29th ACM/IEEE Int. Conf. on Automated Software Engineering*, ser. ASE '14. ACM, 2014, pp. 385–396.

[8] D. Honfi, Z. Micskei, and A. Vörös, "Support and Analysis of Symbolic Execution-based Test Generation, TDK thesis, BME," 2014.

[9] G. Meszaros, *XUnit Test Patterns: Refactoring Test Code*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2006.

[10] Telerik, "JustDecompile Decompilation Engine," 2015. [Online]. Available: https://github.com/telerik/justdecompileengine

[11] N. Tillmann and W. Schulte, "Mock-Object Generation with Behavior," *Proceedings - 21st IEEE/ACM Int. Conf. on Automated Software Engineering, ASE 2006*, pp. 365–366, 2006.

[12] M. R. Marri, T. Xie, N. Tillmann, J. De Halleux, and W. Schulte, "An Empirical Study of Testing File-System-Dependent Software with Mock Objects," *Proc. of the 2009 ICSE Workshop on Automation of Software Test, AST 2009*, pp. 149–153, 2009.

[13] S. J. Galler, A. Maller, and F. Wotawa, "Automatically Extracting Mock Object Behavior from Design by Contract Specification for Test Data Generation," in *Proceedings of the 5th Workshop on Automation of Software Test*. ACM, 2010, pp. 43–50.

[14] M. Islam and C. Csallner, "Dsc+Mock: A Test Case + Mock Class Generator in Support of Coding against Interfaces," in *Proc. of the 8th International Workshop on Dynamic Analysis*. ACM, 2010, pp. 26–31.

[15] B. Pasternak, S. Tyszberowicz, and A. Yehudai, "GenUTest: a Unit Test and Mock Aspect Generation Tool," *International Journal on STTT*, vol. 11, no. 4, pp. 273–290, 2009.

[16] S. Anand, P. Godefroid, and N. Tillmann, "Demand-Driven Compositional Symbolic Execution," in *TACAS*, ser. LNCS. Springer Berlin Heidelberg, 2008, vol. 4963, pp. 367–381.

Sharded Joins for Scalable Incremental Graph Queries

János Maginecz, Gábor Szárnyas
Budapest University of Technology and Economics
Department of Measurement and Information Systems
Email: janos.maginecz@gmail.com, szarnyas@mit.bme.hu

***Abstract*—Model query operations form the basis of model-driven software engineering tools and model transformations. While the last decade brought considerable improvements in distributed storage technologies, known as NoSQL systems, evaluation of graph-like queries on large models requires further research. Unlike typical NoSQL workloads, model queries often include lots of join and complex filtering operations. Thus, the evaluation of such queries on continuously changing data proves to be a challenge for query engines. In this paper, we present INCQUERY-DS, a distributed graph query engine, which utilizes sharding to allow scaling for larger models.**

I. INTRODUCTION

Model-driven software engineering (MDE) plays an important role in the development processes of critical embedded systems. With the dramatic increase in complexity that is also affecting critical embedded systems in recent years, modeling toolchains are facing scalability challenges as the size of design models constantly increases, and automated tool features become more sophisticated. Many scalability issues can be addressed by improving query performance.

Traditional query approaches reevaluate the entire query on every modification, which is expensive for large datasets. In contrast, with incremental query evaluation, the reevaluation is only calculated on parts of the model impacted by the change. This leads to a significant speedup for large, continuously changing data. The Rete algorithm [3] is an incremental algorithm that keeps the partial matches in memory. These matches are stored in nodes that are also the units of computation, i.e. each node performs a relational algebraic operation.

Sharding or *horizontal partitioning* of data is a technique widely used in NoSQL databases and stream processing engines [12]. However, up to our best knowledge, it has not been applied to incremental query engines. In this paper we adopt the idea of sharding to distributed query processing networks.

While sharding mitigates the problem of memory exhaustion, we should also reduce the memory consumption of our tools without performance degradation. The performance of the join operation is crucial in this area, hence we also present the performance comparison of different join algorithms.

This work was partially supported by the MONDO (EU ICT-611125) project and the MTA-BME Lendület 2015 Research Group on Cyber-Physical Systems.

II. PRELIMINARIES

A. Running Example: the Train Benchmark

We use the Train Benchmark [5] to present the core concepts of our approach.[1] The benchmark was designed to measure the efficiency of model queries under a real-world MDE workload. It defines a railway network composed of typical railroad items, including routes, semaphores, and switches (Figure 1).

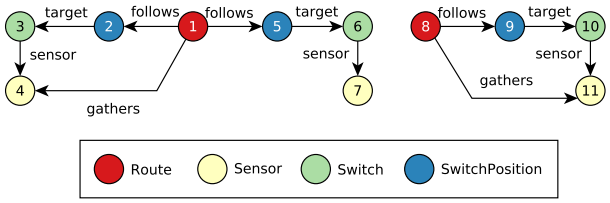


Fig. 1: Train Benchmark example model.

B. Model Validation with Graph Queries

Engineering models can be represented as *typed graphs* with labeled vertices and edges. For example, the edges of the graph in Figure 1 can be represented with the following relations:

- follows (Route, SwitchPosition) : $(1, 2), (1, 5), (8, 9)$
- gathers (Route, Sensor) : $(1, 4), (8, 11)$
- sensor (Switch, Sensor) : $(3, 4), (6, 7), (10, 11)$
- target (SwitchPosition, Switch) : $(2, 3), (5, 6), (9, 10)$

Well-formedness validation constraints are often checked with *graph queries* [2]. The model is queried with graph patterns that search for *violations of the constraint* in the model. The result of a graph query is a set of *tuples*. The RouteSensor constraint requires that all sensors that are associated with a switch that belongs to a route must also be associated directly with the same route. The constrain can be translated to a graph query (in the lower left corner of Figure 2), which looks for sensors that are connected to a switch, but the sensor and the switch are not connected to the same route.

Graph queries can be formalized in relational algebra. Here we only elaborate the join and antijoin operations, as their performance has the greatest effect on query evaluation. The rest of the operations are discussed in [13].

[1]The framework is available as an open-source project at https://github.com/FTSRG/trainbenchmark

- The *join* operation ($\bowtie$) is used to connect relations based on their attributes. The *natural join* operation performs the join based on mutual attributes of the relations.
  Example: the target $\bowtie$ follows $\bowtie$ sensor query selects the matching $\langle$SwitchPosition, Switch, Route, Sensor$\rangle$ tuples.
- The *antijoin* operation ($\triangleright$) is used to express *negative conditions*. Formally, $r \triangleright s = r \setminus \pi_R (r \bowtie s)$.
  Example: the sensor $\triangleright$ gathers query selects the $\langle$Switch, Sensor$\rangle$ pairs not connected to a Route.

Based on these examples, the RouteSensor query can be formalized as:

$$\text{target} \bowtie \text{follows} \bowtie \text{sensor} \triangleright \text{gathers}$$

On the example graph, the result set consist of the tuple $(5, 6, 1, 7)$. This indicates that the model is *not well-formed* w.r.t. the RouteSensor constraint, as there should be a gather edge from node 1 to node 7.

C. Incremental Query Evaluation

Rete [13] is an algorithm for incremental query evaluation. It ensures incrementality by keeping partial matches in memory. This is essentially a *space-time tradeoff*, an approach widely used in computer science (e.g. lookup tables, caching).

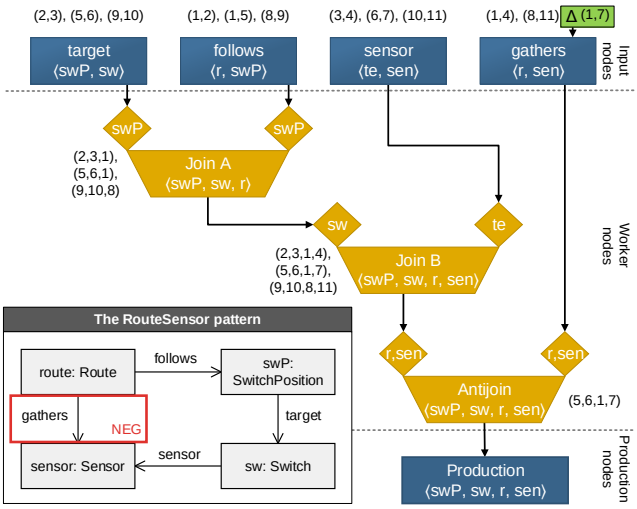


Fig. 2: The RouteSensor pattern and its Rete network.

The Rete algorithm constructs a network of processing nodes consisting of three layers (Figure 2). The partial matches (represented as tuples) are propagated through the network as messages.

- *Input nodes* store the model elements: target, follows, sensor and gathers.
- *Worker nodes* implement relational algebra operators: Join A, Join B and Antijoin.
- *Production nodes* store the results of the query.

If the engineer inserts the missing gathers edge by adding the tuple $(1, 7)$ to the gathers input node (marked with a Δ character in the figure), the Rete network only has to reevaluate the results of *Antijoin* and *Production* nodes.

III. Related Work

EMF-IncQuery is an incremental query engine for models defined in the Eclipse Modeling Framework (EMF). It uses the Rete algorithm for incremental query evaluation [2]. As EMF-IncQuery is a single workstation tool, the memory consumption of the Rete algorithm does not allow it to scale for arbitrarily large models.

Similarly to EMF-IncQuery, IncQuery-D is based on the Rete engine but it was designed from the ground up as a distributed pattern matching system [13]. It allows for using NoSQL databases and triplestores as data sources, which means that the input of the engine can be distributed. IncQuery-D's workflow is similar to its predecessor, but it deploys the Rete nodes over a distributed system.

Drools [10] is a business rule management system that provides a *rule engine* that is capable of checking well-formedness constraints. Drools also uses the Rete algorithm as well to support incremental query evaluation. Rete-based query evaluation is used for processing Linked Data as well. INSTANS [11] uses this algorithm to perform complex event processing on streaming RDF. Diamond [9] also uses a Rete network to evaluate SPARQL queries on RDF data sets.

IV. Overview of the Approach

This section describes the methods used to make IncQuery-DS fast and scalable.

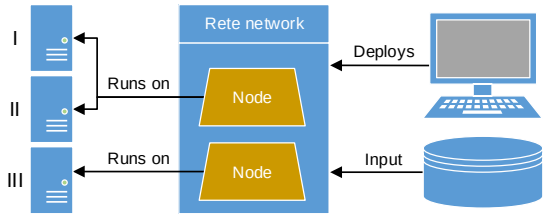


Fig. 3: The architecture of IncQuery-DS.

A. Sharded Rete Algorithm

As a Rete node stores partial matches of the graph, its memory consumption is proportional to the size of the model. This causes *memory exhaustion* for large models. However, it is possible to split a Rete node to multiple node shards. This way a logical node can be distributed across multiple computers, splitting the memory requirements between the shards (Figure 3).

Previous work [14] only focused on distributing the Rete nodes between the machines in the cloud, but did not shard individual Rete nodes in the network. This implies that each Rete node needs to fit in the memory of a single computer, which limits the scalability of the system. For example, in the network for RouteSensor most of the memory is consumed by a single node (Join B), so distributing the Rete network does not allow it to scale for arbitrarily large models. However, sharding allows us to distribute the content of a single Rete network on multiple machines.

To achieve high performance, the computations of a Rete node must be performed based on the contents of a single shard (thus avoiding the communication overhead between the shards). For the join/antijoin nodes this requires tuples with the same join key, from both inputs, to be sent to the same shards.

A sharded layout is shown in Figure 4. Node Join B is split into two shards, Shard 0 and Shard 1, allocated on Host II and Host III. The tuples of Join A, $\{(2,\underline{3},1),(5,\underline{6},1),(9,\underline{10},8)\}$, joined against the tuples of the sensor input node, $\{(\underline{3},4),(\underline{6},7),(\underline{10},11)\}$. The join keys are their second (sw) and first (te) attributes, respectively.

For distributing the tuples, we use two hash functions. First, we map the join key to a number. Consider the simple hash function $h(\langle k_1,k_2,\ldots,k_n\rangle) = 37\sum_i k_i \mod 16$. This produces the following hash values for the keys:

$$h(\langle 3\rangle) = 15, \quad h(\langle 6\rangle) = 14, \quad h(\langle 10\rangle) = 2$$

To shard the tuples, we use another hash function, which simply uses modulo s, where s is the number of shards: $g(x) = x \mod s$. Here, $s = 2$, hence

$$g(h(\langle 3\rangle)) = 1, \quad g(h(\langle 6\rangle)) = 0, \quad g(h(\langle 10\rangle)) = 0$$

Based on the hash values, the tuples with the join key 6 and 10 are processed by Shard 0, while the tuples with the join key 3 are processed by shard 1.

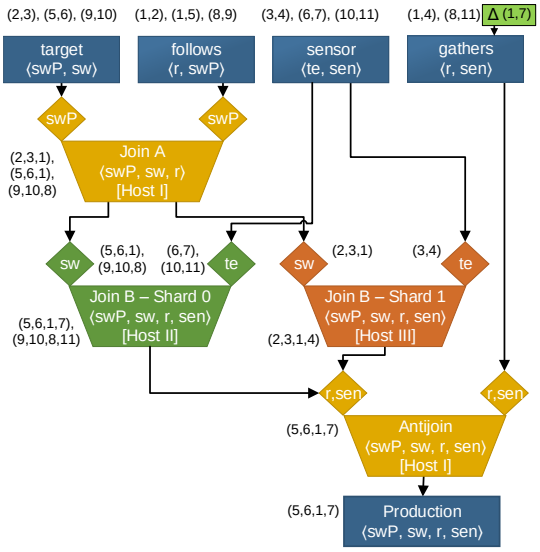


Fig. 4: Layout of the sharded Rete network for RouteSensor.

INCQUERY-DS makes the degree of sharding for each Rete node a separate decision, i.e. some nodes might have many shards, while others may remain unsharded. This greatly affects the performance of the network. Efficient node allocation is out of the scope of this paper, but is discussed in [8].

B. Join node optimization

As mentioned in our previous report [7], different join algorithms and their underlying data structures have a significant effect on the query performance. To elaborate this, we compare the *incremental performance* of the *hash join* and the *sort merge join* algorithm [4]. We implemented both algorithms with both the standard Scala library data structures[2] and a third-party collection framework, GS-Collections[3] (developed by Goldman Sachs Group, Inc).

V. EVALUATION

A. Benchmark environment

The benchmarks were executed on virtual machines with the following setup: 2 cores of an Intel Xeon E5420 processor running at 2.50 GHz, 8 GBs of memory, Ubuntu 14.04 LTS operating system, Oracle JDK 8 runtime with 4 GBs of heap memory, and Gigabit Ethernet network.

B. Benchmark phases

We use the "Repair" scenario of the Train Benchmark. In this scenario, the model is loaded and validated. Next, a subset of the model is transformed and revalidated (Figure 5). This aims to simulate the workload of a user applying quick fixes to the model. The *memory consumption* and *execution time* is recorded for each phase.

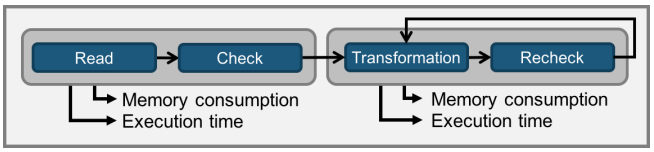


Fig. 5: Phases of the Repair Scenario.

C. Benchmark goals

We benchmarked various aspects of the system.

1) Scalability: To measure the scalability improvements provided by the sharded join algorithm, we executed a benchmark on three machines in four settings:

- As a non-incremental baseline, we used Jena [1], a state-of-the-art RDF-based SPARQL in-memory query engine.
- To compare the scalability of the various degrees of distribution the benchmark measured 3 variants of INCQUERY-DS.
  - The Local variant acts as an incremental baseline, allocating all nodes on a single machine.
  - The Distributed variant allocates each node on separate computer, but does not utilize sharding.
  - The Sharded variant also allocates two nodes on separate computers, but the third node is split into two shards, allocated on different machines.

The transformation change set is indicated with a Δ character in Figure 4. This figure also shows the allocation of the worker nodes in the Sharded variant.

2) Join Algorithm Performance: We compared the performance of join algorithms and collection frameworks. This benchmark was executed on a single machine and only used Join A.

[2]http://docs.scala-lang.org/overviews/collections/overview.html
[3]https://github.com/goldmansachs/gs-collections

D. Benchmark results

Figure 6 and Figure 7 show the results of the benchmarks. In both figures, the x-axis shows the number of triples in the model, while the y-axis shows the time required for the run. Both axes use a logarithmic scale.

1) Scalability: Figure 6 shows the results for repeated evaluations of the RouteSensor query. For large models (5M+ triples), Jena is two orders of magnitude slower than the incremental variants. Compared to the Local variant, the network overhead of Sharded INCQUERY-DS is apparent, but the response time in the "Transformation and Recheck" phase is still within the subsecond range. The Sharded variant handles models twice as large as the Distributed variant. The Distributed variant fails on the largest model, because Join B runs out of memory.

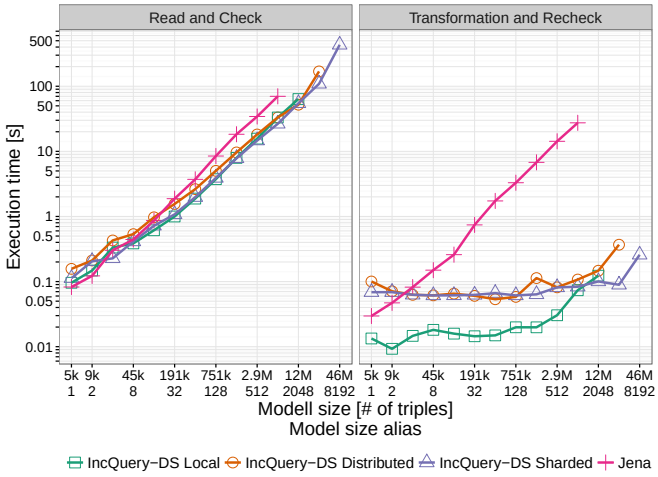


Fig. 6: Scalability of the RouteSensor query with hash join.

2) Join Algorithm Performance: Figure 7 displays the execution times of the different join algorithms with different underlying data structures. The "Check" phase times do not differ significantly, but the implementation using sort merge join are characteristically slower than the hash joins in the "Transformation and Recheck" phase, since the merge join algorithm has to iterate over relevant parts of the data. Comparing the GS and Scala hash joins, we can conclude that the GS variant provides a modest improvement in both scenarios.

Table I shows the memory consumption of each algorithm-implementation pair on the Join A node. The memory consumption of the different algorithms does not differ significantly, but the GS implementations use consistently less memory in every observation.

Join node \ Model size alias	512	1024	2048
GS-HashJoiner	30.0	54.5	103.7
Scala-HashJoiner	36.5	68.5	131.6
GS-MergeJoiner	34.0	63.1	120.5
Scala-MergeJoiner	37.3	69.9	134.0

TABLE I: Memory usage of the Join A node [MB]

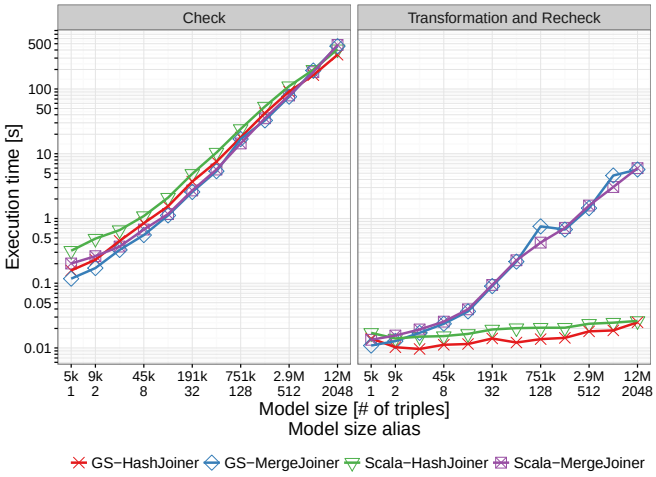


Fig. 7: Comparison of different join algorithms.

VI. CONCLUSION AND FUTURE WORK

In this paper we presented and evaluated a truly scalable incremental query evaluation framework prototype. The results imply that the approach provides high performance for use cases requiring incremental query evaluation, while scaling well for large models.

As future work, we plan to integrate INCQUERY-DS with existing stream processing frameworks, e.g. Kafka [6].

ACKNOWLEDGEMENTS

We want to thank Gábor Bergmann and István Ráth for their continuous support.

REFERENCES

[1] Apache Software Foundation. Apache Jena. https://jena.apache.org/.
[2] G. Bergmann, Á. Horváth, I. Ráth, D. Varró, A. Balogh, Z. Balogh, and A. Ökrös. Incremental evaluation of model queries over EMF models. In *Model Driven Engineering Languages and Systems - 13th International Conference*, pages 76–90. Springer, 2010.
[3] C. Forgy. Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. *Artificial Intelligences*, 19(1):17–37, 1982.
[4] H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database systems - the complete book (2. ed.)*. Pearson Education, 2009.
[5] B. Izsó, G. Szárnyas, and I. Ráth. Train Benchmark. Technical report, Budapest University of Technology and Economics, 2014.
[6] J. Kreps, N. Narkhede, J. Rao, et al. Kafka: A distributed messaging system for log processing. In *Proceedings of the NetDB*, 2011.
[7] J. Maginecz. Scalable incremental graph query evaluation. Student Report, Budapest University of Technology and Economics, 2015.
[8] J. Makai, G. Szárnyas, Á. Horváth, I. Ráth, and D. Varró. Optimization of Incremental Queries in the Cloud. In *CloudMDE*, 2015.
[9] Miranker, Daniel P. et al. Diamond: A SPARQL query engine, for linked data based on the Rete match. *AImWD*, 2012.
[10] Red Hat. Drools. http://www.drools.org/.
[11] M. Rinne. SPARQL update for complex event processing. In *ISWC'12*, volume 7650 of *LNCS*, pages 453–456. 2012.
[12] M. Stonebraker. SQL databases v. NoSQL databases. *Communications of the ACM*, 53(4):10–11, 2010.
[13] G. Szárnyas. Superscalable Modeling. Master's thesis, Budapest University of Technology and Economics, Budapest, 2013.
[14] G. Szárnyas, B. Izsó, I. Ráth, D. Harmath, G. Bergmann, and D. Varró. IncQuery-D: A Distributed Incremental Model Query Framework in the Cloud. In *ACM/IEEE 17th International Conference on Model Driven Engineering Languages and Systems*, pages 653–669. Springer, 2014.

Evaluation of Fault Tolerance Mechanisms with Model Checking

Vince Molnár, István Majzik
Budapest University of Technology and Economics
Department of Measurement and Information Systems
Budapest, Hungary
Email: {molnarv, majzik}@mit.bme.hu

***Abstract*—Failure Mode and Effects Analysis (FMEA) is a systematic technique for failure analysis. It aims to explore the possible failure modes of individual components or subsystems and determine their potential effects at the system level. Applications of FMEA are common in case of hardware and communication failures, but analyzing software failures (SW-FMEA) poses a number of challenges. Failures may originate in permanent software faults commonly called bugs, and their effects can be very subtle and hard to predict, due to the complex nature of programs. Therefore, an automatic method to analyze the potential effects of different types of bugs is desirable. Such a method could be used to automatically build an FMEA report, or to evaluate different failure mitigation techniques based on their effects on the outcome of faults. This paper follows the latter direction, demonstrating the use of a model checking-based automated SW-FMEA approach to assess error detection mechanisms of safety-critical embedded operating systems.**

***Index Terms*—Failure Mode and Effects Analysis, SW-FMEA, model checking, fault tolerance, error detector**

I. Introduction

Safety and in particular the risk of failure is one of the main concerns of safety-critical systems. Certification requires the systematic analysis of potential failures, their causes and effects, and the assessment of risk mitigation techniques used to reduce the chance and the severity of system-level failures.

One of the first systematic techniques for failure analysis was Failure Mode and Effect Analysis (FMEA) [2]. FMEA is often the first step in reliability analysis, as it collects the potential failure modes of subsystems, their causes and their effects on the whole system. Together with criticality analysis (often treated as part of FMEA, sometimes emphasized by the term FMECA), the output of FMEA serves as the basis of other qualitative and quantitative analyses, as well as design decisions regarding risk mitigation techniques.

FMEA is usually applied at the hardware and communication level, where it requires a qualified analyst to collect postulated component failures and identify their effects on other components and the system level. In case of software (SW-FMEA), failure modes originate in different types of programming faults, commonly referred to as bugs. Due to the complex nature of software and the many types of potential bugs, it is much harder to collect failure modes and deduce their potential effects, so an automated mechanism is desired.

This paper presents a way of automated SW-FMEA with the use of executable software models. Assuming a set of predefined fault types (programming errors) and a specification of safe behavior at the system level, the proposed approach applies model checking to generate traces leading from fault activations to states that violate the specification (system-level failures). These traces can be used to understand and demonstrate fault propagation through the system and also as test sequences for the final product.

In addition to automated SW-FMEA, the proposed approach can be used to evaluate the efficiency of fault tolerance and error detection mechanisms. Compared to the baseline of having only the core system model, fault tolerance mechanisms should mask as many faults as possible (reducing the number of fault activations that can lead to a system-level failure), while error detectors should catch the propagating error on as many traces as possible. The evaluation of the latter mechanism is presented on a case study, using a model of the OSEK API specification [1], which is a commonly used interface specification for embedded operating systems.

The paper is structured as follows. Section II introduces the key concepts of FMEA and model checking, then a framework for model checking-based FMEA is outlined in Section III. Applications of the approach are discussed in Section IV, while Section V presents a case study. Section VI provides the concluding remarks and our directions for future work.

II. Background

This section summarizes the main idea of FMEA and in particular SW-FMEA, as well as model checking that is the basis of the approach presented in Section III.

A. Failure Mode and Effects Analysis

FMEA involves *1)* the enumeration of potential failure modes of subsystems, *2)* an inductive reasoning of their effects on different levels of the whole system (called *error propagation*), and *3)* often the deductive analysis of their root causes. The analysis is usually based on a model or specification of a component, as well as historical data and experience with similar components. The result is recorded in an FMEA spreadsheet. Failure modes are then categorized based on criticality, representing the level of chance and the severity of potential consequences. Criticality can prioritize failures, and based on the discovered causes and effects, fault-tolerance or

error detection mechanisms can be designed to mask or detect faults to ensure fail-safe operation of the system.

There are three main concepts related to error propagation. A *failure* is an incorrect system function, i.e., an observable invalid state. An *error* is a latent invalid state that has no observable effects yet. Finally, a *fault* is the cause of a failure, which can be either some kind of defect (physical or design) or the failure of a related subsystem.

During error propagation, an *activated fault* can cause an error, which will turn into a failure once it becomes observable (e.g., by crossing an interface). FMEAs are usually performed on many levels during the design of a system, so a failure of a component is often a fault in another one. FMEA usually assumes that only a single failure mode exists at a time.

Software FMEA: When performing FMEA on software components, failure modes are usually caused by programming (or configuration) errors. The challenge of analyzing them is twofold. First, it is very hard to come up with a realistic set of programming faults (called a *fault model*). The source of bugs is almost always a human, and the most typical faults highly depend on the programming language and the domain as well. Constructing a realistic fault model is therefore even harder in case only a design model is available. Secondly, the effects of a bug is hard to track as it evolves in a complex system.

This paper focuses on the challenge of analyzing the effects of programming faults. The problem of designing proper fault models is not discussed here, we refer to the fact that it is also an important challenge in the field of *mutation-based testing*. For an extensive overview of mutation-based testing and fault models, the reader should refer to [7].

Most of the previous approaches to SW-FMEA build on software testing (e.g., [8]), injecting faults directly into the program code and running a set of tests to see any possible global effects. Model-based SW-FMEA has also been proposed recently [4] based on executable software models, model-level fault injection and simulations. Another approach, similar to the one presented in this paper, uses model checking to detect the violation of the system-level specification in case active faults are present [5]. The common features of these approaches include a predefined set of faults injected into the code or model, a description of system-level failures/hazards, and some sort of execution (either testing, simulation or model checking) to generate traces connecting the first two.

Our approach presented in Section III improves existing model checking-based SW-FMEA by optimizing fault activations and using monitors instead of a formal specification.

B. Model Checking

Model checking is an automated formal verification technique used to verify whether a system satisfies a requirement or not. This is done by systematically (and typically exhaustively) analyzing the states and/or possible behaviors of the system model (i.e., the *state space*). If the specification is violated, model checkers prove it with a *counterexample*.

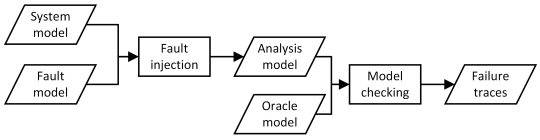


Fig. 1: Overview of the presented approach.

In this work, the tool SPIN was used as a model checker [6]. SPIN is an explicit model checker (using an explicitly stored graph representation of the state space) capable of reachability analysis (is there a reachable "bad" state?) and linear temporal model checking (describing complex temporal behavior). Its strengths include its maturity, the rich set of configuration opportunities and the expressiveness of its input model, given in PROMELA (PROcess MEta LAnguage).

III. Model Checking-based Software FMEA

The approach presented in this paper focuses on the "Effect Analysis" part of FMEA. Assuming a set of possible faults (failure modes) in the software and a characterization of system-level failures, it examines an executable model of the system to generate traces leading to system-level failures.

The process (shown in Figure 1) starts with fault injection, when the input model is transformed into an analysis model containing faults that can be turned on or off. It is assumed that there is an oracle model that allows the detection of system level failures (see Section III-2 for details), so the model checker can analyze the model to check if any fault can cause a system-level hazard. The output is a set of traces that lead from every dangerous fault activation to reachable system-level failures.

1) Fault Injection: The method requires a fault model in terms of the modeling language. Here, a fault model is assumed to be a set of alterations (mutations) that can be applied on the model. The actual alteration is performed by adding a *trigger variable* to activate or deactivate the fault, i.e., with the trigger variable set to *false*, the model should behave correctly, while a value of *true* should cause an erroneous state when the affected part is executed. Note that trigger variables become part of the system as auxiliary state variables.

Using the trigger variables, a number of different fault types can be modeled. First, a fault can be permanent (only nondeterministic $false \rightarrow true$ transitions) or transient (nondeterministic $true \rightarrow false$ transitions are also present). Although in case of software bugs, the faults are usually permanent, it is sometimes useful to have transient faults to simulate the effects of hardware faults as well. Second, it is sometimes desired to restrict the number of faults in the system to at most one, or in some cases at most two.

By injecting the fault activation mechanism into the model, a model checker is free to choose which fault to activate by setting the trigger variables as long as it meets the restrictions.

2) Failure Detection: The traditional approach in model checking is to provide a formal specification of the system. Automated FMEA can then check if the specification still holds in the presence of faults (as described in [5]).

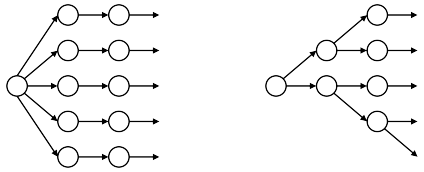


Fig. 2: Shape of the state space with eager and lazy evaluation.

In this work, we suggest an alternative that is closer to a safety engineer's viewpoint. Instead of specifying a failure (or the correct behavior), it is sometimes easier to model a component to detect and signal requirement violations. Such a model can be idealistic (e.g., it may observe every detail of the system or have infinite memory), since its only role is to provide a definition for system-level failures, it does not have to be implemented in the real system. Due to these properties, we will call this idealistic component an *oracle*. Depending on the goals and the domain, an oracle can be a reference model or a more permissive abstract contract.

3) Failure traces: From the extended model and the oracle, the model checker will be able to generate a set of traces leading to system-level failures. From each trace, we can extract the values of the trigger variables (i.e., which faults were activated) and the location of the system-level failure detected by the oracle. If the oracle can classify the failure, this information can also be retrieved.

4) Efficiency and Lazy Evaluation: Model checking is highly sensitive to the size and potential values of the state vector. Unfortunately, adding a set of nondeterministic boolean variables (here the trigger variables) increases the number of potential states exponentially. Moreover, if permanent faults are modeled in such a way that the initial activation is random, the number of initial states immediately blows up exponentially.

In order to avoid the combinatoric explosion, we suggest a "lazy" strategy to evaluate fault activations. Let the trigger variables have ternary values, with the third value being *undefined*, also being the initial value. By injecting additional logic to access the value of trigger variables, it is possible to defer the valuation and have identical states for multiple fault configurations up to an actual fault activation. This effect is illustrated in Figure 2.

IV. Evaluation of Fault Tolerance Mechanisms and Error Detectors

Section III outlined a general approach to model checking-based automated SW-FMEA. In this section, we present two novel applications of the method to evaluate fault tolerance mechanisms and error detectors. The goal is to measure the efficiency of these mechanisms by analyzing what type of faults they can mask or detect, respectively.

For an "absolute" measure, one can use an idealistic oracle (like we suggested in Section III-2) as a baseline and "upper bound" on the efficiency of realistic approaches. In case of error detectors, it is also possible to compute the *relative efficiency* of two solutions, i.e., how much "better" or "worse" is one of them compared to the other.

The measurement setting is the following. In case of error detectors, we first run a check with the oracle (or the first detector) to get the total number of traces leading to observable failures (denoted T as *total*), then we measure the same number (denoted D as *detected*) with the evaluated (or second). In case of fault tolerance mechanisms, both steps use the oracle, with the mechanism coupled with the system in the second step. Efficiency is then defined as follows.

- In case of error detectors, the efficiency is $E = D/T$.
- In case of fault tolerance mechanisms, the efficiency is $E = (T - D)/T$.

Efficiency can also be defined in case of fault types (or failure modes), giving a more detailed picture about the evaluated technique. By obtaining a number describing the efficiency of different approaches, we hope to help design decisions concerning what error detectors and fault tolerance mechanisms to use (possibly in some combination).

V. OSEK API – A Case Study

To demonstrate the merits of the proposed approach, we used the model of the OSEK API [1], a common interface definition for safety-critical embedded operating systems. In a related project[1], an OSEK-compliant real-time operating system targeting the automotive industry had to be certified according to ISO 26262 [3]. The developers of the OS wanted to add fault tolerance and monitoring techniques addressing potential programming errors, both from the side of the OS and client applications. To aid design decisions, we have developed the presented approach to evaluate different solutions still in the modeling phase. For the analysis, we used a model of the OSEK API and a set of test programs (both correct and incorrect) taken from [9] and a set of error detectors with assertions providing the "error signals".

The OSEK API provides a set of interface functions with their syntaxes and also the semantics of the implemented OS primitives. The API defines primitives for task handling and scheduling; resource, interrupt and event handling; semaphores and messaging; as well as times and alarms. For the case study, we used a model describing the Task API, the Resource API and the Event API.

We have modeled two types of error detectors: as the idealistic oracle, a (fault-free) Reference Model that is compared to the state of the OSEK model after each interface call (according to the Master-Checker pattern); as well as a simple Priority Checker that observes only the priorities of scheduled tasks. The Priority Checker can detect if the scheduler violates the priorities, for example by preempting a task to run another one with lower priority.

A. Implementation of Automated SW-FMEA

We have implemented the approach described in Section III based on the model checker SPIN. Fault injection was per-

[1]This work has been partially supported by the CECRIS project, FP7–Marie Curie (IAPP) number 324334.

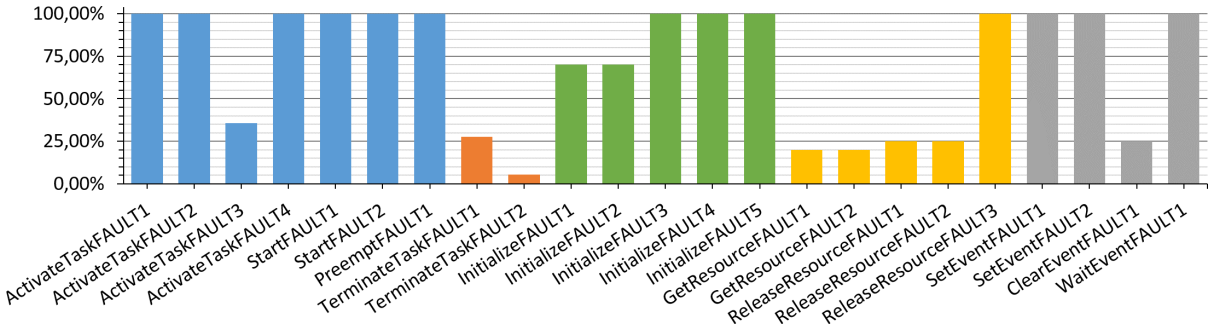


Fig. 3: Efficiency of the Priority Checker compared to the Reference Model

formed by an auxiliary program that parsed the PROMELA model of the OSEK API and altered the code. We used a very simple fault model: each instruction in the model could be removed when activated by a trigger variable. We assumed a single, permanent fault that activates in the initial state.

SPIN was configured to perform a bounded depth-first search optimized for safety checking and enumerating every violating trace. The tool looked for assertion violations (errors detected by the evaluated error detectors) and invalid end states (i.e. deadlocks). Once the model checking finished, the path was replayed to obtain the last (violating) state, containing the values of the trigger variables and the location of the error signal. This information was aggregated for all traces, resulting in the number of violating traces for each different fault-type.

B. Results

Running the analysis with the two detectors showed the relative efficiency of the Priority Checker compared to the more "heavyweight" Reference Model. The diagram in Figure 3 illustrates the efficiency for each fault type (alteration in the API model) separately, also grouping them based on the related API. Although the fault model is artificial, the diagram highlights that the Priority Checker can barely detect faults in the resource handling or task termination primitives, but it is comparable to the Reference Model for most of the faults related to rescheduling (starting tasks and handling events).

In a real world example, analysis of the characteristics of different detectors could help in understanding their efficiency (or coverage) better. In this study, the Reference Model can also be regarded as an idealistic oracle, while something like the Priority Checker can be implemented for an acceptable cost. By knowing the costs of a solution and its characteristics, it should be easier for engineers to find a cost-optimal solution with the highest possible benefits.

VI. Conclusion and Future Work

This paper presented a method for automated SW-FMEA based on model checking, along with a novel idea for applying such approaches in the evaluation of fault-tolerance mechanisms and error detectors.

The main idea of the model checking-based method is to *1)* use model-level fault injection (or model mutations) with trigger variables to augment the system model with switchable faults, then *2)* use formal specification or an oracle model to characterize system-level failures so that *3)* the model checker can generate traces leading from a fault activation to a failure.

Evaluation of fault-tolerance mechanisms and error detectors is based on the notion of (relative) efficiency that describes the number of masked/revealed errors compared to an oracle or another technique (respectively). We hope that this additional piece of information can aid safety-engineers in early design decisions.

The main contribution of this paper is the outline of a general idea. In order to make it applicable, there are a number of further concerns to be considered. First, the fault model for executable software models has a great impact on the validity of the results, so a fine-tuned and validated fault model is necessary. We plan to use completed projects with code-level fault injection to statistically compare the effects of model-level and code-level faults. Secondly, a specific model checking algorithm could inherently optimize the structure of the state space without lazy evaluation injected into the model (see Section III-4). Thirdly, the case study presented here is only in a preliminary phase – modeling other aspects of the OSEK API and additional error detectors or fault-tolerance mechanisms will be necessary to extract meaningful results.

References

[1] Road vehicles – open interface for embedded automotive applications. ISO 17356, 2005.

[2] Potential failure mode and effects analysis in design (design FMEA), potential failure mode and effects analysis in manufacturing and assembly processes (process FMEA). SAE J 1739, 2009.

[3] Road vehicles – functional safety. ISO 26262, 2011.

[4] V. Bonfiglio, L. Montecchi, F. Rossi, P. Lollini, A. Pataricza, and A. Bondavalli. Executable models to support automated software FMEA. In *Proc. High Assurance Systems Engineering*, pages 189–196. IEEE, 2015.

[5] L. Grunske, K. Winter, N. Yatapanage, S. Zafar, and P. A. Lindsay. Experience with fault injection experiments for FMEA. *Software: Practice and Experience*, 41(11):1233–1258, 2011.

[6] G. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley Professional, 2003.

[7] Yue Jia and M. Harman. An analysis and survey of the development of mutation testing. 37(5):649–678, 2011.

[8] Chris Price and Neal Snooke. An automated software FMEA. In *Proc. International System Safety Regional Conference (ISSRC)*, 2008.

[9] H. Zhang, T. Aoki, and Y. Chiba. A SPIN-based approach for checking OSEK/VDX applications. In *Formal Techniques for Safety-Critical Systems*, volume 476 of *CCIS*, pages 239–255. Springer, 2015.

A Low Cost Magnetic-Field Based Indoor Positioning System

Valter Pasku, Alessio De Angelis, Guido De Angelis, Marco Dionigi, Antonio Moschitta, Paolo Carbone
Department of Engineering
University of Perugia
Perugia, Italy
Email: valter.pasku@studenti.unipg.it, guidodeangelis@ieee.org,
{alessio.deangelis, marco.dionigi, antonio.moschitta, paolo.carbone}@unipg.it

***Abstract*—This paper describes the operation principle and experimental results of a low cost positioning system based on low frequency magnetic fields. In particular, the system prototype is realized using low cost microcontroller based boards combined with off-the-shelf components and is tested over an area of approximately 50 m^2. Experimental results show a mean positioning error lower than 0.2 m and a maximum positioning error lower than 0.7 m.**

Keywords—indoor positioning; localization; magnetic fields; resonators.

I. INTRODUCTION

In the last 10 years, the knowledge of the position of different users has become a key factor in the development of several systems and applications. Examples are the Internet of Things (IoT) [1], Location Based Services (LBSs) [2] and Ambient Assisted Living (AAL) [3] applications. While in outdoor environments the Global Positioning System (GPS) solution is a well-established choice, in indoor environments it has a low applicability due to the lack of the satellites visibility. In fact, due to this, an embedded Indoor Messaging System (IMES), based on a GPS receiver [4], has been proposed. It has the goal of seamless indoor-outdoor positioning providing better than GPS accuracies in indoor environments. Moreover, several alternative indoor positioning systems and techniques have been developed. In particular, embedded positioning systems based on Wi-Fi signals and fingerprinting techniques [5] are a common solution. Low positioning performance and high effort database building and maintenance, attribute to this solution a decreasing attention by the scientific community. Systems based on ultrasound sensors offer centimeter level positioning accuracy performance [6], however they are short range and highly affected by non-line-of-sight (NLOS) conditions. Nowadays, systems based on narrow-band radio frequency signals like Bluetooth Low Energy (BLE) [7] are becoming more and more popular. They are built using standard hardware and require low power consumption and low cost. The main drawback is the positioning performance. In general, this class of positioning systems is based on proximity algorithms and received-signal-strength (RSS) measurements, that are affected by multipath phenomena and lead to large positioning errors. In order to improve the performance, a high number of nodes may be required, increasing installation and maintenance costs. Systems based on Ultra Wide Bandwidth (UWB) signals are easily found in the literature [8]. They are characterized by fine resolution time measurements and better penetration properties with respect to the narrow-band solutions. However, obstructions in the direct path, that are common in crowded environments, contribute to the performance degradation. On the other hand, inertial navigation [9] and magnetic field based [10]-[13] positioning systems are not affected or are more resistant to NLOS conditions, respectively. Due to their integrative nature, inertial navigation systems are affected by rapidly growing positioning errors. Instead, magnetic fields based positioning systems usually rely on bulky and expensive electronic instruments and require high power consumption or are characterized by high complexity and computation. Specifically, the magnetic positioning system described in [12], based on three square-shaped low-frequency magnetic transmitters with side dimension of 1 m, and a three axis magnetic receiver, provides 2D and 3D position measurements. The obtained mean positioning error over an experiment area of approximately 30 m × 40 m, is 0.8 m and 0.4 m in the case of 3D and 2D positioning, respectively. The required current consumption for each transmitter is of the order of 7–8 A, and the analog to digital conversion at the receiver has a resolution of 24 bit. The outdoor remote positioning system in [13], which is based on magnetoquasistatic fields, provides 6 DoF measurements of a mobile transmitter by using seven magnetic field receivers. The magnetic transmitter with diameter of 16.5 cm is fed by a class E oscillator with a total output power of 0.56 W and the received signal by each receiver with diameter of 1 m is sampled with a resolution of 16 bit. The system performance is investigated over an outdoor experiment area of 27.43 m × 27.43 m with a mean positioning error of 0.77 m, mean inclination orientation of 9.67° and mean azimuthal orientation error of 2.84°.

In this paper, the authors investigate the performance of a first stage towards an embedded solution of a previously developed magnetic positioning system prototype [10][11], which showed a mean positioning error lower than 0.3 m over an indoor area of 12 m × 15 m and outdoor area of 14 m × 30 m. The required power and current consumption of each transmitter with radius equal to 14 cm is lower than 0.15 W

and 0.02 A respectively. Here, low cost microcontroller based boards have been used instead of high cost electronic instruments improving system portability and decreasing power consumption and cost. A positioning accuracy of the order of tens of centimeters is obtained over an area of approximately 50 m^2. The paper is organized as follows. Section II describes the theoretical basis of the measurement model while Section III provides an overview of the architecture description. In Section IV the experimental results are provided and described and in Section V the conclusions are given.

II. Measurement Model

In order to realize an embedded system solution, usually simplicity and low power requirements may be of fundamental importance. Thus, taking the previous goals in account, the developed magnetic positioning system prototype [10][11] is based on a simple measurement model. In the following, a short summary of the theoretical basics is reported. First, an electromagnetic point of view is considered. The magnetic field generated by a circular coil assumed in the coordinate system origin, in the near field zone, can be modeled as a magnetic dipole [14]

$$\mathbf{B}(x,y,z,t) = \frac{\mu_0}{4\pi}\left[3\mathbf{n}(\mathbf{m}\cdot\mathbf{n})d^{-3} - \mathbf{m}\,d^{-3}\right]e^{-j\omega t} \qquad (1)$$

where μ_0 is the free space magnetic permeability, $\mathbf{n}$ is the observation versor, $\mathbf{m}$ is the coil's magnetic moment, d is the observation distance, ω is the operating frequency and t is the time. The magnetic moment $\mathbf{m}$ is orthogonal to the coil surface and has a module m equal to $m = NIS$, where N is the number of turns in the coil, I is the feeding current and S is the coil's surface. According to Faraday's law of induction, the time varying magnetic field will couple a sensor coil with the field generating coil, inducing a voltage in the former [10][14]. In the case that both sensor coil and field generating coil are oriented in the same way, see Fig. 1(a), the relation between the induced voltage and the distance between the coil's centers, in a bi-logarithmic scale, can be expressed as:

$$\log_{10} V = a - b \log_{10} d \qquad (2)$$

where a is a hardware-configuration dependent constant and b has a nominal value of 3. The interested user can refer to [10] for detailed descriptions. In indoor environments, due to the interaction between the magnetic field and the different objects, the slope of the model (2) can assume different values [10]. Thus, a preliminary system calibration phase, that can be obtained by induced voltage measurements at known distances followed by linear fitting, is required in order to improve system accuracy and robustness [10][11]. During operating conditions, using calibration parameters and induced voltage measurements, ranging can be obtained by inverting (2).

In order to obtain high operating ranges without high power consumption requirements, resonant coils may be used [11]. Considering an equivalent circuit point of view, as in Fig. 1(b), it can be shown that the induced voltage is given by:

$$V \propto V_0 k \sqrt{Q_1 Q_2} \qquad (3)$$

where V_0 is the feeding voltage, k is the coupling factor and Q_1 and Q_2 are the resonator quality factors given by $Q=\omega L/R$, with L and R representing resonator's inductance and resistance, respectively. Thus, using high-Q resonant coils increases the induced voltage at a given distance. Consequently, once the receiver sensitivity is fixed, the operating range can be increased by using high-Q resonators. This is analogue to the antenna gains in a telecommunication system. High-Q resonators can be obtained by increasing the operating frequency, however this will increase the interaction between magnetic fields and surrounding environment [15], decreasing the system robustness. Thus, a tradeoff must be considered. In the following, an operating frequency of approximately 25 kHz will be used, since it provides good robustness and operating ranges up to 30 m or 12 m in line of sight (LOS) and NLOS conditions respectively.

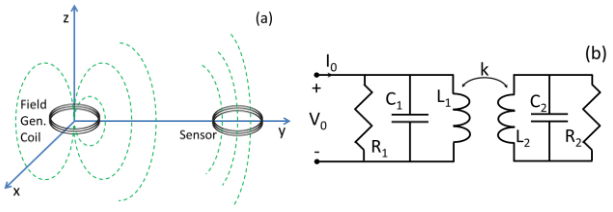


Fig. 1. Representation of the system configuration by considering an electromagnetic (a) and circuit (b) point of view. (a)-The magnetic field generated by a field generating coil at the coordinate system origin will couple with a sensor coil, inducing a voltage. Both the coils are assumed with the same orientation, lying in the *xy* plane. (b)-Circuit representation of the resonant coils assumption.

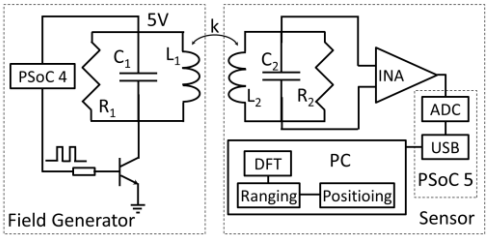


Fig. 2. Schematic of the system architecture. The field generating resonator is driven by a square wave signal provided by a PSoC 4 board. After being amplified by an INA, the output of the sensor coil is digitized by the PSoC 5 on-board ADC, and finally is transmitted through USB to a PC. Distance and position estimation are performed in the PC.

III. Architecture Description

The realized positioning system prototype is based on two types of system nodes, a set of magnetic field generating ones and a sensor node. A schematic representation of the system architecture, which is realized using off-the-shelf components, is shown in Fig. 2.

In particular, the field generating nodes are composed by a resonator and a driving circuit realized with a microcontroller based board, the Programmable System on Chip (PSoC) CY8CKIT-049 of the PSoC 4 family by Cypress. Each resonator is formed by the parallel connection between a circular coil with 20 turns of radius equal to 15 cm showing a nominal inductance of 128 µH and a lumped capacitor of 330 nF. The realized resonators have a nominal resonant frequency of 24.5 kHz and a quality factor of 12. The PSoC 4 board is programmed in order to provide a square wave signal with a frequency near to the resonance. Due to the resonator's band pass behavior, only the first harmonic of the square wave will have a significant contribution to the generated magnetic field, with the higher harmonics leading to a negligible effect. The required driving current has a maximum value of approximately 20 mA rms, leading to a magnetic field intensity

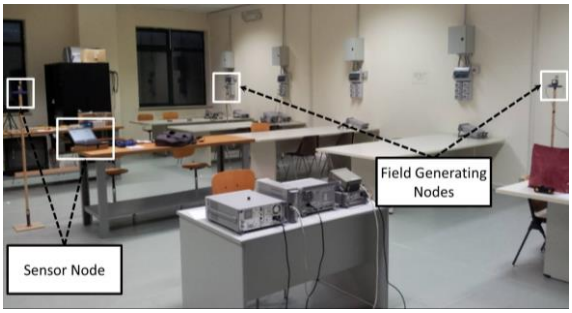


Fig. 3. Picture of the experiment environment. Two of the field generating nodes and the sensor node are shown.

of approximately one order of magnitude lower than the international regulations [10]. In order to avoid synchronization requirements that increase the system complexity, each field generating node is characterized by a particular frequency that is related with the known position of the node and identifies it at the receiver side.

The sensor node is composed by the same resonator as the field generating one, and also includes all the electronics for signal conditioning, acquisition and processing. Specifically, the induced voltage, which is self-filtered by the resonator, is first amplified by an instrumentation amplifier (INA), the integrated circuit AD8421BRZ by Analog Devices, and then acquired by a CY8CKIT-050 PSoC 5LP microcontroller based board. The digitalized signal, obtained by using a 12 bit ADC configured with a sampling rate equal to 75.65 kSa/s is transferred to a PC through a USB connection with the PSoC 5LP board. Then, the available signal, that comprises the contribution of all the field generating nodes plus noise, with a length of 30 kSa, after a flat-top windowing process, is used as an input to a DFT based algorithm for amplitude estimation. In particular, the usage of slightly different operating frequencies allows simultaneous amplitude estimation without time synchronization. Inverting the measurement model (2) and using the estimated amplitudes, range measurements are obtained. The Euclidean distance between a transmitting node with coordinates (x_i y_i) and the mobile node at a position with coordinates (x y) is given by $d_i = \sqrt{(x - x_i)^2 + (y - y_i)^2}$ while the corresponding estimated range is represented by $\hat{d}_i$. The measurement of the sensor node position relative to the field generating nodes is obtained by minimizing in a least squares sense the non-linear cost function given by:

$$E = \left[\hat{\boldsymbol{\Delta}} - \boldsymbol{\Delta}_{x,y}\right]^T \left[\hat{\boldsymbol{\Delta}} - \boldsymbol{\Delta}_{x,y}\right] \qquad (4)$$

where $\hat{\boldsymbol{\Delta}} = \left[\hat{d}_1 \;\; \hat{d}_2 \;\; \hat{d}_3 \;\; \hat{d}_4\right]^T$ contains the measured distances, $\boldsymbol{\Delta}_{x,y} = [d_1 \; d_2 \; d_3 \; d_4]$ contains the Euclidean distance between the field generating nodes and the position with coordinates (x y) and T is the transpose operator. Future developments include the realization of a fully embedded solution of the realized prototype. In particular, we are focusing on real time on-board amplitude estimation, range and position measurement without the usage of a PC.

IV. Experimental Results

In order to test the performance of the developed system prototype it was setup in a laboratory environment, as shown in Fig. 3, over an area of 7.2 m × 7.2 m. The system nodes were operating in LOS conditions, however several disturbing objects like metal cabinets, tables, chairs and electronic instruments were inside the experiment area. First, using four field generating nodes which do not require any time synchronization, static repeated measurements were performed in a set of nine known position points where real-time positioning of the sensor node was obtained. Then, position estimation in a dynamic scenario was considered. The results are shown in Figs. 4-6. In particular, in Fig. 4 the system deployment and the repeated measurement results are shown where the black squares represent the field generating node positions, the red diamonds the true positions and the blue asterisks the estimated positions. The measurement points were taken at the edges, at the center and at the side centers of a square with dimensions 3 m × 3 m. The true positions were manually surveyed with an estimated accuracy of the order of a couple of centimeters, and for each of them 150 position measurements were considered. The mean positioning error was evaluated for each point and a maximum value lower than 0.3 m was obtained. Further information regarding positioning error characteristics is provided by Fig. 5. In particular, the boxplot of the positioning error of all the considered points is shown. The positioning error shows a unimodal distribution and a maximum error of approximately 0.4 m, except a few outliers showing an error lower than 0.7 m. The mean error of all the measurements is approximately 0.17 m.

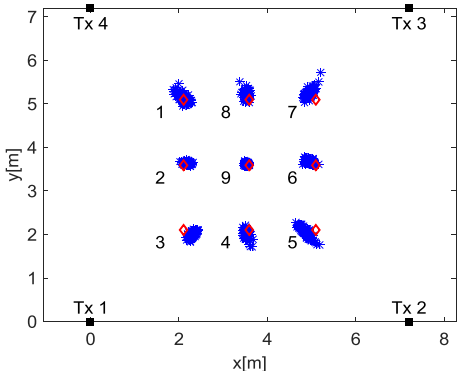


Fig. 4. Positioning results of 150 repeated measurements for each point over nine static points. The true positions are represented by the red diamonds while the estimated positions by the blue asterisks. Field generating resonator positions are shown by black squares.

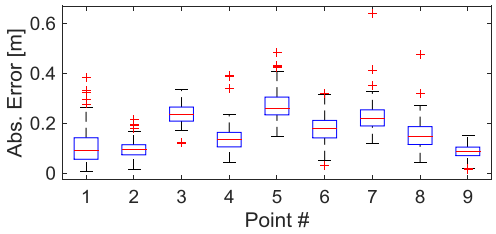


Fig. 5. Boxplot representation of the positioning error over the repeated measurements represented in Fig. 3. The central line of each box is the median error while the edges of each box represent the first and third quartile. Top and bottom ends of the dashed black lines are the maximum and minimum error with the red plus symbols representing outliers.

System simplicity, intended as measurement model leading to low computational cost, is obtained by using geometrical assumptions and limitations, such as operation in a planar and with equally oriented coils scenario. Since in general this can restrict the system applicability, measurements in a dynamic

scenario where the geometrical assumptions are not always satisfied are a key element for the system performance characterization. Two dynamic measurements were performed. In particular, a user followed a square reference path defined by the static measurement points, carrying the magnetic sensor node with himself. In both cases, the starting point was the number 1, shown by the black cross in Fig. 6. The first trajectory was the counterclockwise path starting in point 1 and ending in point 8, repeated three times. The second path was defined by two repetitions of the previous path, followed by the sequence 9-4-3-2-1-8-9. The real-time position estimation provided by the system is shown in Fig. 6. In particular, the blue dot-dashed line represents the first trajectory and the blue dashed line the second one. Estimated stop positions are represented by the magenta circle and the plus symbol. It can be clearly seen that the estimated positions are in good agreement with the followed paths. In general, it has been observed that misalignments or coplanarity deviations of the order of 10° only introduce a positioning error of the order of 5-10 cm [10][11]. System performance in outdoor scenarios was also tested, obtaining a similar performance even in larger coverage areas.

In general magnetic fields may be distorted by metals and/or different in-band noise sources in the environment. Some non-ideal effects, such as constant amplitude in-band interferences, can be mitigated by calibration, others may require lower operation frequency and/or higher SNR which can be obtained with higher power consumption. We have noted that the proposed method has a good robustness in typical indoor/outdoor environments since substantial part of the noise is filtered out by the band pass behavior of the resonators. Instead, in-band noise may be filtered out by considering non-linear coding and correlation techniques.

V. Conclusions

In this paper, a partially embedded version of a previously developed low cost magnetic positioning system is presented. A short overview of theoretical basics and system architecture is given and new obtained experimental results are discussed. Repeated measurements over an area of approximately 50 m^2 show a mean positioning error lower than 0.2 m with a maximum positioning error lower than 0.7 m. Moreover, measurements over two dynamic trajectories show good agreement with the reference path.

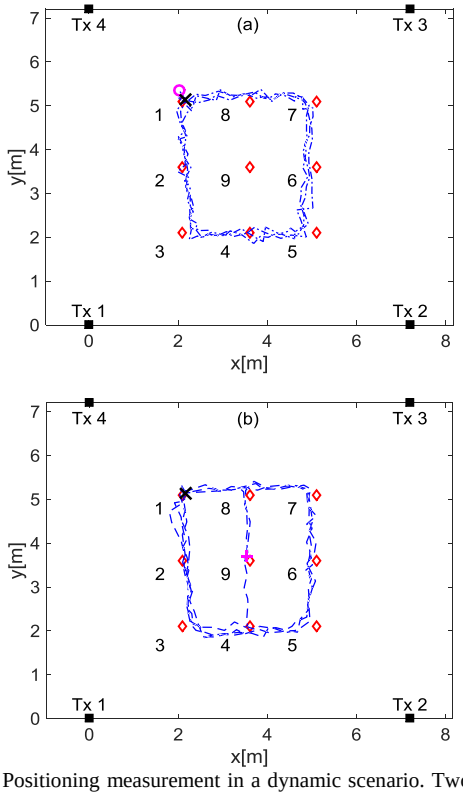


Fig. 6. Positioning measurement in a dynamic scenario. Two trajectories were followed with the first (a) consisting in the square defined by the external red diamonds and the second (b) consisting in two rectangles defined by the red diamonds. The results are represented by the blue dot-dashed line and the blue dashed line respectively. The start position is shown by the black cross and the stop positions by the magenta circle and magenta plus respectively.

References

[1] Perera, C.; Liu, C.H.; Jayawardena, S.; Min Chen, "A Survey on Internet of Things From Industrial Market Perspective," *IEEE Access*, vol.2, pp.1660-1679, 2014.

[2] Schiller, J., Voisard, A. "Location-based services," *Elsevier*, 2004.

[3] Barsocchi, P.; Chessa, S.; Furfari, F.; Potorti, F., "Evaluating Ambient Assisted Living Solutions: The Localization Competition," *IEEE Pervasive Comput.*, vol.12, no.4, pp.72-79, Oct.-Dec. 2013.

[4] IMES, [Online]. Available: http://gpsworld.com/wirelessindoor-positioningopening-up-indoors-11603/, May 2011.

[5] He, S.; Chan, S.-H.G., "Wi-Fi Fingerprint-based Indoor Positioning: Recent Advances and Comparisons," *IEEE Commun. Surveys Tuts., 2015.*

[6] De Angelis, A.; Moschitta, A.; Carbone, P.; Calderini, M.; Neri, S.; Borgna, R.; Peppucci, M., "Design and Characterization of a Portable Ultrasonic Indoor 3-D Positioning System," *IEEE Trans. Instrum. Meas.*, vol.64, no.10, pp.2616-2625, Oct. 2015.

[7] iBeacon, [Online]. Available: http://www.ibeacon.com/, Retrieved 2015.

[8] Conti, A.; Guerra, M.; Dardari, D.; Decarli, N.; Win, M.Z., "Network Experimentation for Cooperative Localization," *IEEE J. Sel. Areas Commun.*, vol.30, no.2, pp.467-475, February 2012.

[9] Nilsson, J. O.; Gupta, Amit K; Händel, P., "Foot-mounted inertial navigation made easy," *International Conference on Indoor Positioning and Indoor Navigation (IPIN), 2014*, pp.24-29, 27-30 Oct. 2014.

[10] Pasku, V., De Angelis, A., Dionigi, M., De Angelis, G., Moschitta, A., Carbone, P., "A Positioning System Based on Low Frequency Magnetic Fields," *IEEE Trans. Ind. Electron*, (**in press**), 2015.

[11] De Angelis, G., Pasku, V., De Angelis, A., Dionigi, M., Mongiardo, M., Moschitta, A., Carbone, P., "An Indoor AC Magnetic Positioning System," *IEEE Trans. Instrum. Meas.*, vol. 64, no. 5, pp. 1275-1283, May 2015.

[12] Sheinker A., Ginzburg B., Salomonski N., Frumkis L., and Kaplan B. Z., "Remote tracking of a magnetic receiver using low frequency beacons", *Measurement Science and Technology*, Vol. 25, 2014, 105101.

[13] Arumugam, D.D., Griffin, J.D., Stancil, D.D., Ricketts, D.S., "Three-Dimensional Position and Orientation Measurements using Magnetoquasistatic Fields and Complex Image Theory," *IEEE Antennas Propag. Mag.*, vol. 56, no. 1, pp. 160-173, Feb. 2014.

[14] Balanis, C. A., *Antenna theory: analysis and design*, Vol. 1, John Wiley & Sons, 2005.

[15] Pasku, V.; De Angelis, A.; Dionigi, M.; Moschitta, A.; De Angelis, G.; Carbone, P., "Analysis of the sensitivity of AC magnetic ranging systems to environmental configurations," *2015 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, pp.1877-1882, 11-14 May 2015.

Novel Technique for Radiation Dose Visualization in Large Space

Martins Piksis
Institute of Biomedicine and Nanotechnology
Riga Technical University
Riga, Latvia
martins.piksis@liepajasslimnica.lv

***Abstract*—**From the point of view of a medical physicist the creation of 3D visualizations to illustrate radiation dose distribution within a room or building is a very useful tool for radiation protection planning. This study describes a novel technique for 3D visualization of radiation field, where the source modeling is completely separated from the visualization process. Basically it uses an input file from pre-simulated sources that actually significantly improves the computing time compared to the real-time simulations. Visualization module read this information stored in coordinate form with a representative value at each point, and constructs elemental shapes at specified coordinates. All shapes are associated with an intensity value, related to color scale. The shapes are colorized and then, enhanced with transparency effects.

Keywords—3D dose visualization, radiation field, radiation protection planning

I. INTRODUCTION

Radiation protection planning, is very important aspect of occupational health. There are several comercial computer programs in the global market, which are capable to display ionizing radiation fields within 3D environment, but mostly they have high price and very poor user interface. Radiation protection planning could be greatly enhanced by providing staff with a simple and easy to use tool to make source simulations and generate 3D visualizations.

The aim of this study is to create and develop a novel method and software for 3D visualization of radiation fields in large space. This study sought to create the 3D visualization method that any potential user could emulate and adapt for any of a variety of purposes. Furthermore this software could be a useful tool for generation of 3D visualizations for augmented reality applications.

II. CONCEPT OF METHOD

The generation process of 3D radiation field can be divided into several phases as shown in Figure 1.

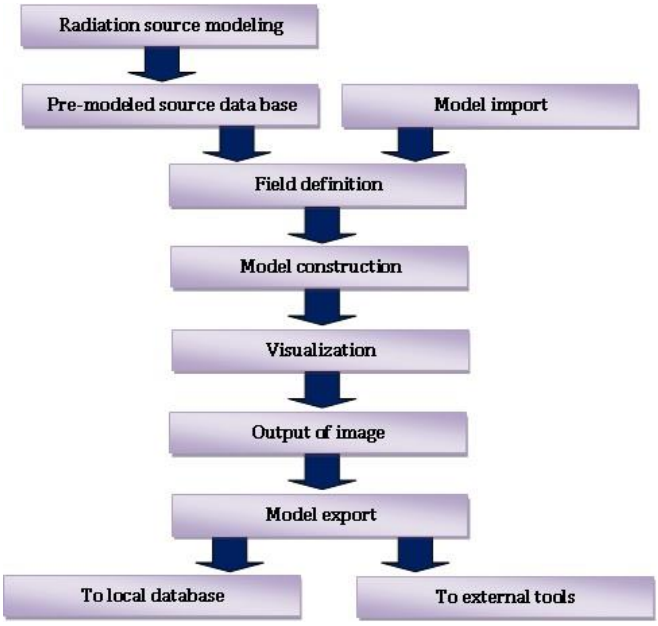


Figure 1. Program concept flowchart

A. Source modeling computational software approach

A computational field model is based on Monte Carlo N-Particle (MCNP) transport code [1]. If a nuclear physics code is able to keep a tally in multiple locations simultaneously, and provide output data in a format that is similar to (or could be made similar to via post processing) that in Table I., then the output from that code is compatible with the model construction process. In general there are no other factors that would limit the users choice of MC code package selection. The modeling technique was tested with such packages as GATE and PRIMO and in both cases satisfactory results were obtained. Post-simulation data is stored in a local database in *.txt format.

Table I. Example data format compatible with Ruby code

(x-coordinate)	(y-coordinate)	(z-coordinate)	(intensity) [%]*
5	**5**	**6**	**4.10**
6	**4**	**6**	**3.80**

** intensity values are expressed as a percentage of the maximum dose*

B. Field definition

The first step is defining of a radiation field. A radiation field is a term in medical physics which most commonly refers to the particle fluence, and also often the energy distribution of ionizing radiation within a medium, volume or space. The key feature of a field is a quantifiable trait, such as the rate of particles entering each volume unit of space that varies throughout all environment.

C. Model construction

The radiation field model construction process is based on iterative, point by point technique. Building a model from many small parts potentially may propose more errors during construction process, but alternatively the construction method is conceptually very easy to understand and adapt for specific projects and programs. Besides, utilization of fundamentally simple concepts maximizes the compatibility of this method with various 3D modeling software packages. The only requirements for target 3D modeling software package is that the software:

1) allows the automation of construction actions,
2) allows transparency effects to be applied.

D. Interacting with a visualization

The final stage of the construction process takes place when the modeled field is used for its visualization. In this stage, viewpoints are setup for their eventual output as static images. The viewing process is entirely reliant on the construction process, as it limits which types of programs may open and view a constructed model. The assumption that the primary users will not be 3D modeling experts, requires that this interaction process be as user friendly as possible.

E. Output of images

Through the interaction process briefly described in previous section, it is necessary to respect a user requirement to collect and output images of the modeled field for inclusion into presentation material, reports or other media forms. The final consideration in the development of this method was that the end technique must make the process of generating this material very easy.

F. Model portability to other 3D software (model export)

Portability of a constructed model to alternative software is considered a desirable trait in this research. Once a field model has been created, its usefulness is directly related to the number of different analytical and visual applications available for user to view and analyze a model. This is primarily concerned with the data format of the 3D model. If a 3D model is stored in an openly documented and available 3D format such as the COLLADA [2] format, it will be possible to use many different 3D viewers. In addition to being compatible with a wide array of modeling software due to no licensing costs, the COLLADA format is based on the Extensible Markup Language (XML). This allows a COLLADA formatted file to be directly opened with a simple text editor program and properties of the file can be edited directly via the text editor. Visualization script was designed using RUBY [3] programming language.

III. KEY CHALLENGES

A. Model visibility

Visibility is a very important issue which needs to be considered very carefully. For a radiation field to be completely visible a computer model must allow a viewer to observe both internal and external details, like looking through a foggy window. This is an absolute requirement to allow all of the internal details within a field to be seen from external viewpoints, otherwise they would be obstructed by the outer layer of information. These details could be something such as a change of dose rate within a localized area, or depending on the type of field being modeled, it could be a change in local particle fluence quantities or other relevant factors. The visibility of non-radiation models (such as background) was also considered an equally important requirement.

B. Model navigation

Navigation refers to how the end user will move around a modeled field, so that the user is able to analyze a model from as many viewpoints as possible. The controls to move a viewpoint in an environment need to be very intuitive in usage. How a user establishes multiple perspectives for a scene and manipulates a model plays a key role in the overall user friendliness of this software.

C. Model colors

Determining appropriate colors to be used in model is the next step needs to be addressed. If models are made with non intuitive colors, obtaining information from model become much more difficult. If a model uses too many colors it could also be difficult to review and understand the results. A set of recommendations concerning the use of color within a visualization will be required at some level to assist users in the creation of models.

D. Model limits

A model limit is a reference to where a radiation field model should be constructed and where it should not. Here is the potential for data excesses particularly for large and complex models. Reduction of data abundance (and computing time) is possible by limiting the model to only the specific section (ROI) needed for a given scenario.

IV. METHOD SUMMARY

A field is devided into a set of finite elements with each element containing a series of bounds, an intensity value and a central coordinate. Each element is considered to act as a single representation of an intensity value for a field within the local bounds of that element. These elements are geometrically simple shapes such as cubes or boxes. These individual elements can be thought of as a physical representation of a volumetric pixel (voxel). Voxels can be used to represent data in a three-dimensional space as they contain both a physical location and a value at that location. Using basic shapes simplifies the arrangement of these elements into a single model where all the elements can be fitted together so their boundaries do not overlap each other. Theoretically an intensity value is not limited to a single type

of information (e.g., dose rate); any type of information could be visualized using this method.

The method proposed for building and modeling a radiation field can be summarized as follows:

1) a data set containing (x_n, y_n, z_n, V_n) is taken (where x_n, y_n and z_n represent coordinates, and V_n represents a value at those coordinates).

The process requires that x_n, y_n, z_n, values be at a fixed distance apart to establish a fundamental element size for that model *(e.g., if they have values at 1m, 2m, 3m,...Nm, in all directions, this process will establish that each (x_n+1) is equal to (x_n+1m) and the fundamental element size is a cube of 1m x 1m x 1m)*

2) a script inside the 3D modeling program is being used for reading those values one at a time and constructs an element at each location [4]. This script includes a scale where the V value is assessed and each element is coloured based on its value.

a) the program is opened and the script is runing to calculate each (x_n, y_n, z_n, V_n) and a shape is built (centered at the coordinate or other reference point)

b) based on the V_n value, that new object (volume) is given a colour, material, or whatever the term the program uses to define the appearance of an element

3) this process then repeats until a shape has been built at all of the locations specified in the data file.

a) during the construction process different ranges of associations can be assigned to values of V. I.e.: if V is: 5> V > 3, then color = light blue which means any time a shape is built, and the V value is less than 5 but greater than 3, a color value of 'light blue' will be assigned

b) all entities within the same range will share the same color, or material property. They require transparency to be added to complete the visual effect.

Visualization of point source sample data shown in Figure 2. and Figure 3.

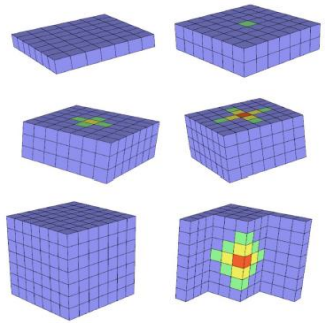


Fig. 2 Example model of point source

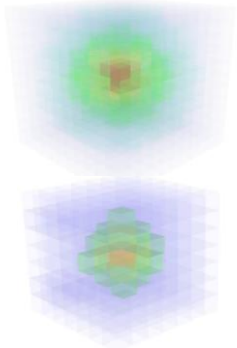


Fig. 3 Example model with transparency effect applied

V. Results

Program accuracy was evaluated in the following stages:

1) assessment of the radiation source Monte Carlo model accuracy relative to the measurements in water phantom,

2) visualized dose model accuracy relative to the $KERMA_{AIR}$ measurements – point dose in software compared with measured dose at specified coordinate in room.

For evaluation purposes a linear accelerator Varian Clinac iX head model was created. Monte Carlo simulations was accomplished for 6MV beam at the field sizes of 5x5 cm; 10x10 cm; 15x15 cm; 20x20 cm. Results verified relative to the measurement in IBA Blue Phantom.

Comparsion of simulated and mesured %PDD is shown in Figure 4.

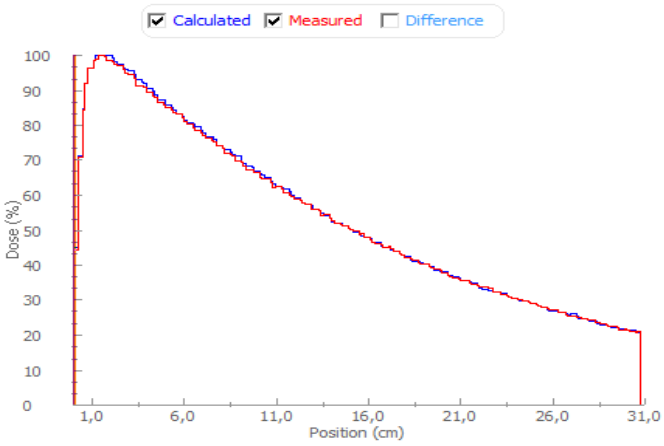


Fig. 4. Comparsion of %PDD for field size 10x10cm.

Similar evaluation was also conducted for dose profiles in X and Y planes.

Results are presented in Figure 5. and Figure 6., where blue curve represents calculated (simulated) dose profile but red curve is related to measured dose profile in water phantom.

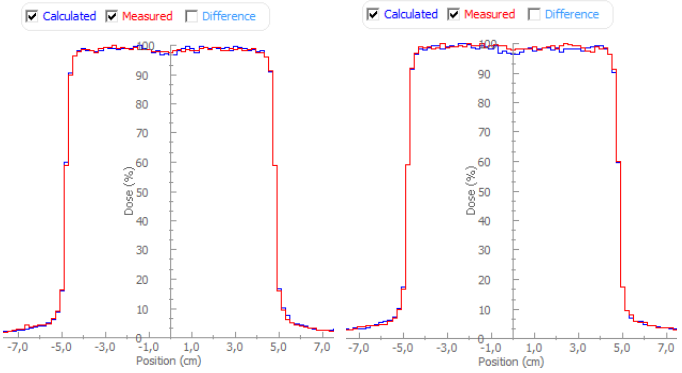


Figure 5. X plane dose profile comparsion, 10x10cm

Figure 6. Y plane dose profile comparsion, 10x10cm

Established MC linac head model accuracy assessed as appropriate for the continuation of the experiment.

For further evaluation of the program was created the linac bunker 3D model, with radiation source (linac) inside, as shown in Figure 7.

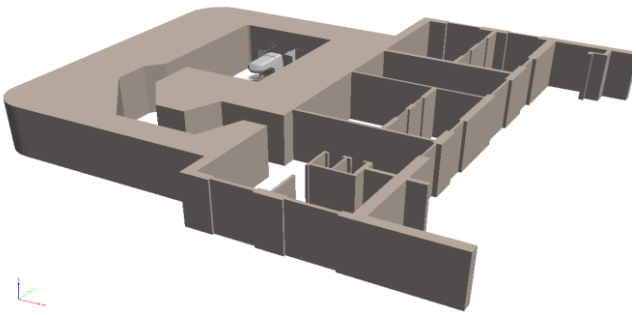


Fig. 7. Varian Clinac iX bunker 3D model, data taken from Liepaja Regional Hospital, scale 1:1

Radiation source data was taken from simulations mentioned above. After completion of the visualization, point doses from model was taken for 46 points and compared with measured dose under the same conditions. Example dose visualization is shown in Figure 8.

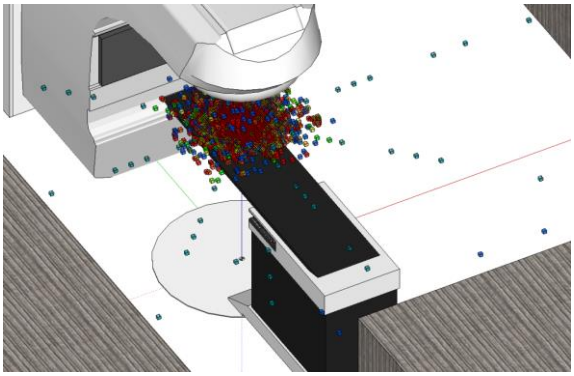


Fig. 8. Experimental radiation field visualization for Varian Clinac iX, 6MV, field 40x40cm.

Results obtained vary between 3.54% and 21.09% in relation to the measured dose. In addition, there is a trend that the error increases with increasing distance from isocenter. It could be explained by the fact that the precision of measurement is less at lower dose rate. The numerical values are partly summarized in the Table II. and Figure 9.

Table II.

X [cm]	Y [cm]	Z [cm]	Δ%
0	0	100	3.54
50	50	100	8.32
300	300	100	21.09

Numerical values of measured dose error according to distance.

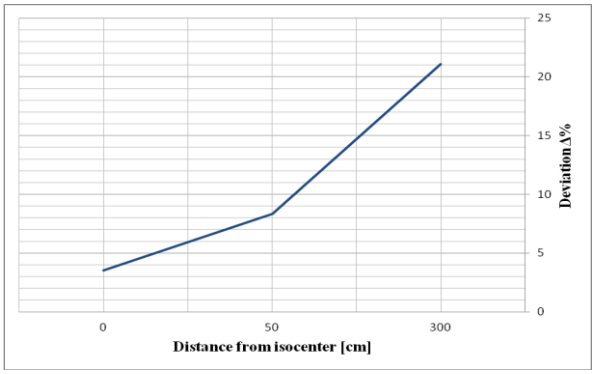


Fig. 9. Measured dose error trend according to distance from isocenter. Breaking point at X 50cm; Y 50cm most likely was caused by use of unsuitable measuring equipment.

VI. CONCLUSIONS

In this study, a novel methodology for the display of 3D radiation fields was developed. New approach was formulated which focused on keeping the field definition process separate from the modeling process to maximize potential definition techniques.

The types of expected issues associated with 3D radiation field visualizations were discussed and analyzed. Overall design requirements for this type of program development were established and eventually shown to have been achieved. The software product obtained in this study, of course, require improvements and adjustments, but generally it has been demonstrated that it is able to operate for its intended purposes. This is a unique case, when this type of modeling method was applied to radiotherapy radiation source modeling and field visualization in three-dimensional space.

Further work is planned severely to work on improvement of the user interface and functionality. As the final result is expected to gain full-fledged software product, which can be used in radiation protection planning.

Acknowledgment

Special Thanks to Mr. Joseph Chaput (IAEA), for sharing ideas and experience, and Mr. Dan Rathbun (PlugIn Store), for the help on the program coding, debuging and practical recommendations.

References

[1] Monte Carlo Team, Monte Carlo N-Particle Transport Code, Version 5 Volume 1.

[2] https://www.khronos.org/collada/, Website last accessed on 12.10.2015

[3] https://www.ruby-lang.org/en/, Website last accessed on 12.10.2015

[4] Joseph G. Chaput A Generic Methodology For The 3-Dimensional Visualization Of Radiation Fields, University of Ontario Institute of Technology, Oshawa, Canada, 2010

Swarm Intelligence Meets Rule-Based Design Space Exploration

Alexandra Anna Sólyom, András Szabolcs Nagy
Department of Measurement and Information Systems
Budapest University of Technology and Economics
Email: solyomalexandraanna@gmail.com, nagya@mit.bme.hu

***Abstract*—In model-driven development design artefacts (e.g. source code and system configuration) are automatically generated from models. For example, a distributed computer system, which consists of multiple different hardware and software elements, can be effectively captured by an appropriate graph-like model, which can be used to generate configuration. Several modelling problems can be automatically traced back to optimization problems, such as finding the most cost effective, reliable or efficient allocation software components to hardware components. However, finding optimal solution for such systems is a major challenge, because (1) existing approaches usually operates over vectors while the problem at hand is defined by graphs; (2) besides the model, the configuration steps may also have to be optimized; and (3) for the best results, optimization techniques should be adapted to the actual domain. In this paper, we propose to integrate the bee colony optimization technique with rule-based design space exploration to solve multi-objective optimization problems in a configurable and extensible way.**

I. Introduction

Design Space Exploration (DSE) is a method for finding various system designs at design or even at runtime, which satisfy given structural and numerical constraints. Besides satisfying these constraints, DSE searches for an optimal or nearly optimal solution.

Model-Driven Rule-Based DSE operates over the model. It starts from an initial model and evolves it in each iteration with use of graph transformation rules, until it reaches one or more constraints satisfying model states. One of the advantages of this approach is that it also provides a sequence of transformations as a solution besides the design candidate itself. Furthermore, this approach is easy to integrate with model-driven development [1].

Graph transformation rules consists of two main parts [2], a graph pattern and an operation. The graph pattern defines locations of applicable transformations, through finding pattern-matching parts of the graph, while the operation determines the possible operations on these subgraphs, using previously given schemas.

Model-driven rule-based DSE can solve multi-objective optimization [3] problems. Best solution of such problems is often non-trivial. There can be more than one equally good solutions, because we have more objectives, which could be contradicting to each other. E.g. the optimization objectives of safety and cost are often conflicting, as improving safety may lead to an increased cost.

Swarm intelligence is an effective heuristic method, for finding a good solution in reasonable time. It is an adaptation of successful natural survival strategies, such as foraging of ants, bees and birds while they are looking for food sources. A common feature among swarm intelligent methods is the simplicity of participating units and the communication between them. The most used swarm-intelligent-based search algorithms are the Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO) and Bee Colony Algorithm (BCA). We found the BCA the most promising candidate for initial investigations as it is the most flexible from the three.

Section II gives insight how the multi-objective rule-based DSE works, and it also introduces the most important concepts needed to understand DSE and Multi-Objective DSE (MODSE). Section III describes swarm-intelligence-based algorithms especially the Bee Colony Algorithm. In section IV, we present our approach to solve DSE problems. Finally, section V concludes the paper.

II. Multi-Objective DSE

Models have two main types, the *metamodel* and the *instance model*. While metamodels describe the structure of models, instance models give the exact description of them. In our case, metamodels define the acceptable structures, which in most cases enable a wide variety of models. DSE-used input models belong to instance models, and they are defined in an unambiguous way [4].

As an example, consider computers and processes in a distributed system, where the metamodel defines the possible elements – computers and processes – and possible connections between them, while the instance model gives the exact number of computers and processes and how they are connected.

A graph transformation rule defines how an instance model can be modified. A transformation rule (Figure 1) consists of two sides: left hand side (LHS) is a constraint, which defines the condition and gives context to the rule while the right hand side (RHS) specifies the operation on the model. Left hand side is given by a *graph pattern*, which consists of constraints on types, connectedness and attributes. A graph pattern has a *match*, when a subgraph in the given graph has the exact structure as the pattern. A graph pattern can have multiple matches on a model [1].

As the graph pattern can have multiple matches, each rule may have multiple activations, and in most cases it is undefined, which rule should be applied. When an activation is applied, the graph structure is modified by the transformation.

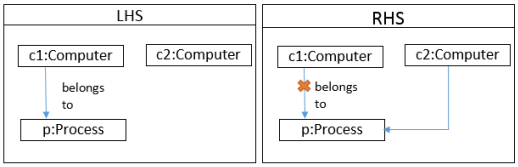


Fig. 1. Graph transformation rule for process reallocation

Well-formedness (WF) constraints (also known as design rules or consistency rules) complement metamodels with additional restrictions that have to be satisfied by a valid instance model (in our case, functional architecture model). Such constraints can also be defined by query languages such as graph patterns or OCL invariants. Ill-formedness constraints capture ill-formed model structures and are disallowed to have a match in a valid model.

For instance in Figure 2, there is a design rule, that every process has to belong to a computer. In this case, in Figure 3 there is a well-formed instance model (a) according to this rule, while in (b) there is an instance model, that is ill-formed. In the ill-formed model there is one match of the ill-formedness constraint.

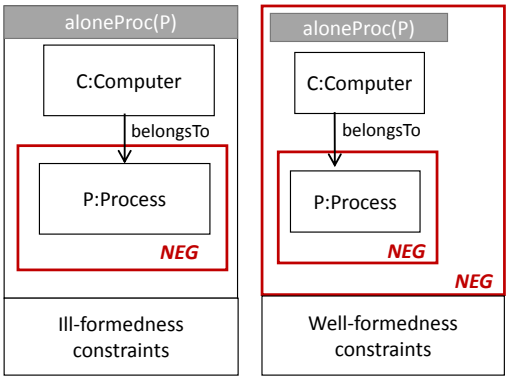


Fig. 2. Structural constraints by graph patterns

A *DSE problem* requires three input parameters: 1) an initial model, 2) a set of graph transformation rules and 3) a set of *goal constraints* captured by graph patterns. A solution of a DSE problem is a sequence of rule applications, which reaches a goal model state that satisfies all the goal constraints. These solutions are found by exploring the search space (or design space), through executing graph transformations according to an exploration strategy.

Multi-objective DSE (MODSE) incorporates objectives that express the quality of a solution. Structural (well-formedness) constraints can be also leveraged to an objective by measuring the degree of constraint violation. These objectives are either to minimize or maximize.

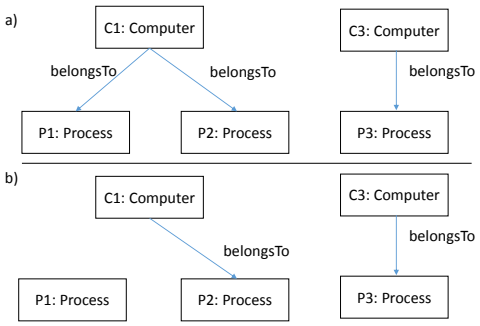


Fig. 3. Example of well-formed (a) and ill-formed instance model (b)

Objectives can be defined on both the trajectory or the model itself. While trajectory objectives measure the quality of the rule application sequence such as number or cost of operations, model specific objectives usually incorporates extra-functional objectives such as performance and reliability.

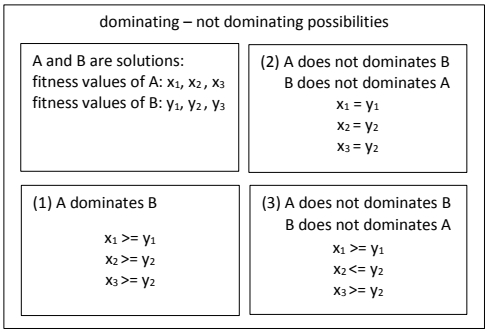


Fig. 4. Domination details x and y values are the fitness values of the solutions

While in a single-objective context solutions are easy to compare to each other, measuring and ranking (evaluating) in multi objective setting is not always obvious. For instance, if there is a computer system that has to be optimized, it is unclear whether three times faster or two times cheaper computers are the better option. It depends mostly on other aspects (size of the company, exact task, etc.), so for the same problem both can be good solutions. Therefore, the *domination function* is used in our DSE implementation to distinguish between solutions and find the best one. An example for domination can be seen in Figure 4. There are two functions, which can be the objective values of two solutions, e.g. cost, response time or safety. A solution dominates another, if at least one objective value (fitness value) is higher than the others and all other values are higher or equal. As a consequence, ordering is unambiguous, a single best solution usually cannot be determined. Instead, a Pareto front is defined, which contains all the ”best” solutions. If a solution belongs to the Pareto front, then none of it’s parameters can be increased without decreasing other parameters. In consequence, all solutions in the Pareto front dominate all other solutions, which are not

part of the Pareto front.

III. Swarm Intelligence Background

Swarm intelligence algorithms are based on modelling living groups, which successfully accomplish specific tasks, like ants, wolves or bees [5]. In these situations, a lonely animal could not survive on his own, though the whole group can. These methods are always heuristic and not aiming to find the best solution as the animals neither do it, but to find a good-enough solution in a reasonable time. Common in these packs is that an individual follows simple rules during the procedure while communicating few information to others. Such techniques can be often used for complex optimization problems, because they have good scalability and flexibility [6].

Some of the well-known swarm algorithms are the Particle Swarm Optimization (PSO), Bee Colony Algorithm (BCA) and Ant Colony Optimization (ACO). We have chosen the BCA for our initial experiments as 1) PSO is originally designed for continuous problem domains and rule-based DSE is a discrete optimization problem, 2) BCA seemed more flexible than ACO in terms of adapting guided local search exploration strategies such as hill climbing.

Bee Colony Algorithm is an often used swarm-intelligence-base algorithm, which attempts to reproduce nectar-searching methods of bee colonies. Normally, bees look for nectar in two phases. In the first phase they look for flower patches where nectar can be found. If a bee founds a patch, it goes back and performs the waggle-dance, which is a communication form between bees. Waggle-dance describes the size of the found patch and the route to it. Depending on the goodness (size, available nectar) of found patches a number of bees go out to look for the food on this patch. Then they come back and tell again how much more nectar is there.

Input parameters of the bee algorithm are: 1) the search space (problem representation, neighbourhood function), 2) the stopping criteria and 3) the size of bee population (n).

The BCA depicted in Figure 5 consists of three main phases:

- scouting phase,
- evaluation phase, and
- collection phase.

These n bees are divided into two groups. One group *(neighbourhood bees)* explores the found patches and continue to map them further, while the other group *(scout bees)* is sent out to look for new ones. Neighbourhood bees can be seen as a local search in possible optimum places while scout bees help to skip from local minimum or maximum places, and switch to a better surrounding. In the scouting phase, the first population of bees is initialized and sent out randomly to collect information. During scouting phase all n bees are scout bees which means, that they randomly explore the search space into different directions. In this phase it is important to avoid generation of similar trajectories to sample the search space in as many directions as possible. When each bee has returned then comes the evaluation phase, when patch ranking is determined, and stop condition is evaluated. Patch ranking helps to decide that which patches are worth for further exploration. If the stop criteria is fulfilled, then the algorithm can be stopped, and the best patches are selected for output. If the stop criteria is unsatisfied then we enter the loop on the right side of Figure 5. In this loop, the best collected patches are selected, and then the collection phase is started. In the collection phase, neighbourhood bees are sent out to the selected patches, and if there are more bees left (from the initial n) then these are sent out as scout bees to search for new patches.

The concrete ratio of scout and neighbourhood bees rely mostly on measuring methods [7]. In our approach, users can select the exact number of bees.

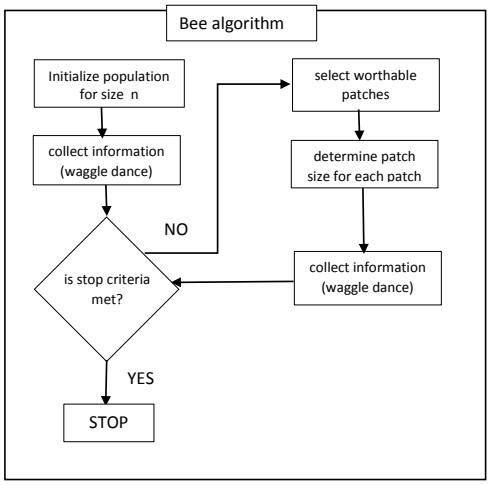


Fig. 5. Algorithm of beestrategy

IV. The Proposed Approach

The aim of our work is to use swarm-intelligence-based algorithms (namely the bee colony algorithm) for multi-objective design space exploration. While the basic challenges such as solution encoding and objective encoding were solved by Abdeen et al. in [3], adapting the bee algorithm has several other challenges, such as:

1) What is the best strategy for scout bees?
2) What is the best strategy for selecting patches for the next iteration (evaluation strategy)?
3) What is the best strategy for the neighbourhood bees?

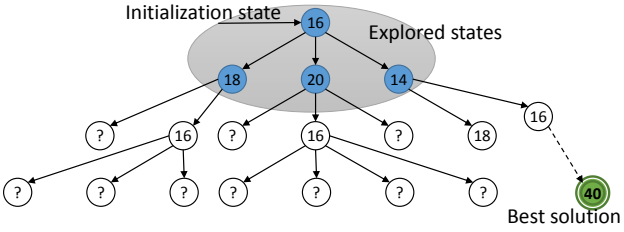


Fig. 6. A possible search space of the bee algorithm

A. Exploration Strategies for Scout Bees

The aim of the scout bees is to generate new solutions that are far enough from each other as well as previously found solutions to prevent the algorithm to stuck in local optimum. Hence, scout bees should use relatively high randomness.

While traditional approaches represent solutions as integer arrays and it is straightforward to generate such solutions, in our approach, the solutions are represented as a sequence of rule applications. We have two options for finding random nodes. The first one is to search nodes from already found patches. The advantage of this option is that it can cut down the search time by some steps, though there are situations in which scouts cannot reach every part of the graph and fail to improve the solution. For example in Figure 6, if the exploration found the three blue states with fitness values 18, 20 and 14, and the scout bees only start from the best ones, 18 and 20, then the exploration will miss the best solution depicted with green.

The other option is to search from the initial model, and go randomly to patches. Then the algorithm is able to find the green solution, though it needs more time. In this case it is hard to reach patches that are further from the initial model, and more likely to find solutions near to the initial state. The good reason behind this is that we are looking for sorted solutions. On the other hand, for scout bees it is hard to decide which direction to follow, because in this situation to reach some of the solutions they have to go through the blue states as well. As a result, we have to search more, possibly all states, which helps to find good solutions, but time-consuming. Another problem is that we have to store quite many additional data about each of the bees and their movements to avoid infinite loops. For instance, which states were good, how many times bees explored it, which other states were reached from them. If we do not store this information, then the bees can iterate through a loop, where each state is contained in the same Pareto front.

B. Evaluation Strategies

The most important decision when choosing an evaluation strategy is whether only the best or some of the worse patches should remain, and to do it in every iteration, or they should be set out only after a while. In some implementation, it is possible to sort out the wrong patches, though in our case it will not be a good idea, because the above-described hill climbing effect would come into sight. We use non-dominating sorting, which allows higher freedom for selecting the correct solutions. Non-dominating sorting means, that the algorithm separates solutions into groups according to their domination levels (number of fronts, which dominate them). In our approach, it is also modifiable how many worse solutions should be taken into consideration.

C. Exploration Strategies for Neighbourhood Bees

Neighbourhood search can be seen as a local search, which aims to discover the surrounding of a patch. It can be a hill climbing style strategy, but it gives an upper limit to the number of bees ordered to a patch. It can be a random strategy, but it is really similar to the random search in the first phase, so it has to be a combination of these two. In our approach, there are more possible strategies from which bees can choose. Some of them are similar to random search and some of them are more like hill climbing strategies with little modifications. However, each of them have some random factor to minimize the possibility of parallel-running-bees collision, which would not be a problem, but it involves unnecessary steps. Some of the usable implementations are:

- *Hill Climbing*: first, it evaluates all the neighbourhood states. Then, it finds dominating ones and randomly selects one of them, if we have enough dominated state in our list. If not, it can choose a non-dominating one that helps to avoid local minimum.
- *Simulated Annealing*: initially it steps randomly, and as time passes, the possibility of choosing a bad transformation decreases. At the end it acts like a the hill climbing algorithm.
- *Depth-First Search*: it goes through all the states till a given depth. It searches the solutions semi-randomly.

V. Conclusion and Future Work

In this paper, we proposed to integrate the bee colony optimization strategy as an exploration strategy with constrained multi-objective rule-based design space exploration. We also analysed the advantages and disadvantages of using different algorithmic configurations of the bee exploration strategy.

As for future work, we would like to measure and evaluate the effectiveness of the approach on a wide range of configurations and compare it with other exploration strategies, such as genetic algorithm and guided local search.

Acknowledgment

This work was partially supported by the MTA-BME Lendület 2015 Research Group on Cyber-Physical Systems.

References

[1] Á. Hegedüs, Á. Horváth, and D. Varró, "A model-driven framework for guided design space exploration," *Automated Software Engineering*, pp. 1–38, 2014.

[2] G. Bergmann, I. Ráth, T. Szabó, P. Torrini, and D. Varró, "Incremental pattern matching for the efficient computation of transitive closures," in *sInternational Conference on Graph Transformation*, 2012.

[3] H. Abdeen, D. Varró, H. Sahraoui, A. S. Nagy, Á. Hegedüs, Á. Horváth, and C. Debreceni, "Multi-objective optimization in rule-based design space exploration," in *International Conference on Automated Software Engineering (ASE)*, 2014.

[4] O. Semeráth, Á. Barta, Á. Horváth, Z. Szatmári, and D. Varró, "Formal validation of domain-specific languages with derived features and well-formedness constraints," *Software & Systems Modeling*, pp. 1–36, 2015.

[5] D. Karaboga, B. Gorkemli, C. Ozturk, and N. Karaboga, "A comprehensive survey: artificial bee colony (abc) algorithm and applications," *Artificial Intelligence Review*, vol. 42, no. 1, pp. 21–57, 2014.

[6] D. Karaboga and B. Akay, "A comparative study of artificial bee colony algorithm," *Applied Mathematics and Computation*, vol. 214, no. 1, pp. 108–132, 2009.

[7] D. Pham and A. Ghanbarzadeh, "Multi-objective optimisation using the bees algorithm," in *Proceedings of IPROMS*, 2007.

Formal Modeling of Real-Time Systems with Data Processing

Tamás Tóth* and István Majzik†
Budapest University of Technology and Economics,
Department of Measurement and Information Systems,
Fault Tolerant Systems Research Group
Email: *totht@mit.bme.hu, †majzik@mit.bme.hu

***Abstract*—The behavior of practical safety critical systems usually combines real-time behavior with structured data flow. To ensure correctness of such systems, both aspects have to be modeled and formally verified. Time related behavior can be efficiently modeled and analyzed in terms of timed automata. At the same time, program verification techniques like abstract interpretation and software model checking can efficiently handle data flow. In this paper, we describe a simple formalism that is able to model both aspects of such systems and enables the combination of formal verification techniques for real-time systems and software. We also outline a straightforward method for building efficient verifiers for the formalism based on the combination of analyses for the respective aspects.**

I. Introduction

Ensuring the correctness of safety critical systems using formal verification is a challenging task as it requires formal modeling of the system in question, as well as the application of formal analysis techniques. Usually, the behavior of practical safety critical systems exhibits both real-time aspects (e.g. switching to an error state after a certain amount of time has passed since the last event occurred) and data flow (e.g. branching on the value of a program variable or initializing a loop counter).

Time-related behavior can be conveniently modeled in terms of timed automata [1]. Model checkers for timed automata like UPPAAL [2] can efficiently verify models using dedicated data structures that represent abstractions over real-valued clock variables [3]. Usually, data variables are handled by encoding the data flow into the control flow [2], which only admits variables of finite domains, or alternatively by using a logical encoding [4]–[11] and then performing model checking by calling to decision procedures. In the latter case, the information about time-related behavior becomes implicit and efficiency depends mostly on the underlying solver.

On the other hand, state-of-the-art program verifiers [12] are designed to handle complex data flow, described in terms of a control flow automaton, and often use abstraction-refinement techniques [13] to handle variables of possibly infinite domains. However, they are not directly capable of verifying timed systems.

*This work was partially supported by Gedeon Richter's Talentum Foundation (Gyömrői út 19-21, 1103 Budapest, Hungary).

†This work was partially supported by the ARTEMIS JU and the Hungarian National Research, Development and Innovation Fund in the frame of the R5-COP project.

In this paper, to enable integration of verification techniques used in real-time verification and program verification, we define a formalism, *Timed Control Flow Automata* (TCFA), that is an extension of *Control Flow Automata* (CFA) used in program verification, with notions of *Timed Automata* (TA), the prominent formalism of real-time verification. Its main advantage is that it represents both data flow and timing explicitly and in a way that is similar to the original formalisms, thus enables the application of analyses that fit to the respective aspects. We define the syntax and semantics of the formalism, and describe how it relates to CFAs and TAs. Furthermore, we outline a simple method for combining analyses for the two formalisms to build efficient verifiers for TCFAs.

II. Background and Notations

In this section, we describe the notations used in the paper.

A. Types

Let Type denote a set of types and Dom a mapping from types to their semantic domains. We assume $\{\mathbf{bool}, \mathbf{int}, \mathbf{real}\} \subseteq \mathit{Type}$ such that $\mathit{Dom}(\mathbf{bool}) = \mathbb{B}$, $\mathit{Dom}(\mathbf{int}) = \mathbb{Z}$ and $\mathit{Dom}(\mathbf{real}) = \mathbb{R}$.

B. Variables

Let Var be a set of program variables. Variables have types, expressed as function $\mathit{type} : \mathit{Var} \to \mathit{Type}$. We abbreviate $\mathit{Dom}(\mathit{type}(v))$ by $\mathit{Dom}(v)$. The set of variables of type $\tau \in \mathit{Type}$ is denoted by $\mathit{Var}(\tau) = \{v \in \mathit{Var} \mid \mathit{type}(v) = \tau\}$.

C. Expressions

Let Expr be a set of well-typed expressions over Var. An expressions can contain program variables $v \in \mathit{Var}$, logical connectives ($\mathbf{true}$, $\mathbf{false}$, $\neg$, $\vee$, $\wedge$, $\rightarrow$, $\leftrightarrow$), quantifiers ($\forall x : \tau.\varphi$, $\exists x : \tau.\varphi$) and logical variables, interpreted function symbols (e.g. 0, $+$, $\cdot$), interpreted predicate symbols (e.g. $\doteq$, $<$, $\leq$), uninterpreted function and predicate symbols, and type constructors and accessors in case the type system supports complex data types. Given an expression $e \in \mathit{Expr}$ and a type $\tau \in \mathit{Type}$, we denote by $e : \tau$ iff e has type τ. Naturally, $v : \tau$ iff $\mathit{type}(v) = \tau$ for all variables $v \in \mathit{Var}$ and types $\tau \in \mathit{Type}$. The set of formulas is denoted by $\mathit{Form} = \{\varphi \in \mathit{Expr} \mid \varphi : \mathbf{bool}\}$.

D. States

A concrete data state $\mathcal{S} \in \mathit{State}$ is a mapping from variables to values such that $\mathcal{S}(x) \in \mathit{Dom}(x)$ for all $x \in \mathit{Var}$. We also extend this notion to arbitrary expressions. For a state $\mathcal{S} \in \mathit{State}$ and formula $\varphi \in \mathit{Form}$, we denote by $\mathcal{S} \models \varphi$ iff $\mathcal{S}(\varphi) = 1$.

E. Statements

Let Stmt denote the set of statements. Although our formalization admits arbitrary structured statements, for the sake of simplicity we assume that statements are of the form

$$s ::= [\varphi] \mid v := e \mid \mathbf{havoc}\, v \mid s\,;\, s$$

where $v \in \mathit{Var}$, $e \in \mathit{Expr}$ and $\varphi \in \mathit{Form}$. Statement $[\varphi]$ is an **assume** statement, $v := e$ is an assignment of e to v, $\mathbf{havoc}\, v$ is an assignment of an arbitrary value of a suitable type to v, and $s\,;\, s$ is a sequential statement.

The semantics of statements can be expressed by the (not necessarily total) semantic function $\mathit{Succ} : \mathit{State} \times \mathit{Stmt} \to \mathcal{P}(\mathit{State})$ that assigns to a state $\mathcal{S} \in \mathit{State}$ and a statement $s \in \mathit{Stmt}$ a set of successor states $\mathit{Succ}(\mathcal{S}, s)$. It can be defined as

- $\{\mathcal{S}\}$ if $s = [\varphi]$ and $\mathcal{S} \models \varphi$
- $\emptyset$ if $s = [\varphi]$ and $\mathcal{S} \not\models \varphi$
- $\{\mathcal{S}' \in \mathit{State} \mid \mathcal{S}' = \mathcal{S}[v \leftarrow \mathcal{S}(e)]\}$ if $s = v := e$
- $\{\mathcal{S}' \in \mathit{State} \mid \mathcal{S}' = \mathcal{S}[v \leftarrow x] \text{ for some } x \in \mathit{Dom}(v)\}$ if $s = \mathbf{havoc}\, v$
- $\{\mathcal{S}'' \in \mathit{State} \mid \mathcal{S}' \in \mathit{Succ}(\mathcal{S}, s_1) \text{ and } \mathcal{S}'' \in \mathit{Succ}(\mathcal{S}', s_2) \text{ for some } \mathcal{S}' \in \mathit{State}\}$ if $s = s_1\,;\, s_2$

F. Timed Automata

Timed automata [1] is a widely used formalism for modeling real-time systems. A TA is a tuple $(\mathit{Loc}, \mathit{Clock}, \hookrightarrow, \mathit{Inv}, \ell_0)$ where

- Loc is a finite set of locations,
- Clock is a finite set of clock variables.
- $\hookrightarrow \subseteq \mathit{Loc} \times \mathit{ClockConstr} \times \mathcal{P}(\mathit{Clock}) \times \mathit{Loc}$ is a set of transitions where for $(\ell, g, R, \ell') \in \hookrightarrow$, g is a guard and R is a set containing clocks to be reset,
- $\mathit{Inv} : \mathit{Loc} \to \mathit{ClockConstr}$ is a function that maps to each location an invariant condition over clocks, and
- $\ell_0 \in \mathit{Loc}$ is the initial location.

Here, $\mathit{ClockConstr}$ denotes the set of clock constraints, that is, formulas of the form $x_i \sim 0$ and $x_i - x_j \sim c$ where $x_i, x_j \in \mathit{Clock}$, $\sim \in \{<, \leq, \doteq\}$ and c is an integer literal.

The operational semantics of a TA can be defined as a labeled transition system $(S, \mathit{Act}, \to, I)$ where

- $S = \mathit{Loc} \times \mathit{State}$ is the set of states,
- $I = \{\ell_0\} \times \{\mathcal{S} \in \mathit{State} \mid \mathcal{S}(x) = 0 \text{ for all } x \in \mathit{Clock} \text{ and } \mathcal{S} \models \mathit{Inv}(\ell_0)\}$ is the set of initial states,
- $\mathit{Act} = \mathbb{R}_{\geq 0} \cup \{\alpha\}$, where α denotes discrete transitions,
- and a transition $t \in \to$ of the transition relation $\to \subseteq S \times \mathit{Act} \times S$ is either a delay transition that increases all clocks with a value $\delta \geq 0$:

$$\frac{\ell \in \mathit{Loc} \quad \delta \geq 0 \quad \mathcal{S}' = \mathit{Delay}(\mathcal{S}, \delta) \quad \mathcal{S}' \models \mathit{Inv}(\ell)}{(\ell, \mathcal{S}) \xrightarrow{\delta} (\ell, \mathcal{S}')}$$

or a discrete transition:

$$\frac{\ell \xhookrightarrow{g,R} \ell' \quad \mathcal{S} \models g \quad \mathcal{S}' = \mathit{Reset}(\mathcal{S}, R) \quad \mathcal{S}' \models \mathit{Inv}(\ell')}{(\ell, \mathcal{S}) \xrightarrow{\alpha} (\ell', \mathcal{S}')}$$

Here, $\mathit{Delay} : \mathit{State} \times \mathbb{R}_{\geq 0} \to \mathit{State}$ assigns to a state $\mathcal{S} \in \mathit{State}$ and a real number $\delta \geq 0$ a state $\mathit{Delay}(\mathcal{S}, \delta)$ such that

$$\mathit{Delay}(\mathcal{S}, \delta)(v) = \begin{cases} \mathcal{S}(v) + \delta & \text{if } v \in \mathit{Clock} \\ \mathcal{S}(v) & \text{otherwise} \end{cases}$$

Moreover, $\mathit{Reset}(\mathcal{S}, R)$ models the effect of resetting clocks in R to 0 in state $\mathcal{S} \in \mathit{State}$:

$$\mathit{Reset}(\mathcal{S}, R)(v) = \begin{cases} 0 & \text{if } v \in R \\ \mathcal{S}(v) & \text{otherwise} \end{cases}$$

G. Control Flow Automata

In program analysis, programs are modeled in terms of control flow automata. Syntactically, a CFA is a tuple $(\mathit{Loc}, \mathit{Var}, \hookrightarrow, \ell_0)$ where

- Loc is a finite set of program locations,
- Var is a set of program variables,
- $\hookrightarrow \subseteq \mathit{Loc} \times \mathit{Stmt} \times \mathit{Loc}$ is a set of control flow edges, and
- $\ell_0 \in \mathit{Loc}$ is the initial location.

The operational semantics of a CFA then can be conveniently expressed in terms of a labeled transition system $(S, \mathit{Act}, \to, I)$ where

- $S = \mathit{Loc} \times \mathit{State}$ is the set of states,
- $I = \{\ell_0\} \times \mathit{State}$ is the set of initial states,
- $\mathit{Act} = \mathit{Stmt}$,
- and the transition relation $\to \subseteq S \times \mathit{Act} \times S$ is defined by the rule

$$\frac{\ell \xhookrightarrow{s} \ell' \quad \mathcal{S}' \in \mathit{Succ}(\mathcal{S}, s)}{(\ell, \mathcal{S}) \xrightarrow{s} (\ell', \mathcal{S}')}$$

H. Abstract Semantics

To ensure efficiency or termination, modern model checkers and program analyzers check abstractions of systems, expressed in terms of abstract domains. An abstract domain is a triple $(S, \mathcal{E}, \gamma)$ where

- S is the set of concrete states,
- $\mathcal{E} = (E, \top, \bot, \sqsubseteq, \sqcup)$ is a semi-lattice over the set of abstract states E with a top element $\top \in E$, a bottom element $\bot \in E$, a preorder $\sqsubseteq \subseteq E \times E$ and a join operator $\sqcup : E \times E \to E$, and
- $\gamma : E \to \mathcal{P}(S)$ is the concretization function that assigns to each abstract state the set of concrete states it represents.

Given a transition system $(S, Act, \rightarrow, I)$ for the concrete semantics, the abstract semantics w.r.t. $\mathcal{E}$ and γ can be expressed as a transition system $(E, Act, \rightsquigarrow, \gamma(I))$. For soundness of the analysis, the following properties must hold:

- $\gamma(\top) = S$ and $\gamma(\bot) = \emptyset$,
- $\gamma(e_1) \cup \gamma(e_2) \subseteq \gamma(e_1 \sqcup e_2)$ for all $e_1, e_2 \in E$, and
- $\bigcup_{s \in \gamma(e)} \{s' \in S \mid s \xrightarrow{\alpha} s'\} \subseteq \bigcup_{e \overset{\alpha}{\rightsquigarrow} e'} \gamma(e')$ for all $e \in E$ and $\alpha \in Act$.

The abstract transition relation $\rightsquigarrow \subseteq E \times Act \times E$ is also called a transfer relation.

A verifier can then analyze the system by exploring the abstract state space and applying abstraction refinement [13] in case of a spurious error path that cannot be simulated according to the concrete semantics.

III. Timed Control Flow Automata

To extend CFAs with timed behavior, we assume $\mathbf{clock} \in \mathit{Type}$ for a distinguished type $\mathbf{clock}$ such that $\mathit{Dom}(\mathbf{clock}) = \mathbb{R}_{\geq 0}$. This enables modeling of a clock variable as a regular program variable of type $\mathbf{clock}$. In his context, $\mathit{Clock} = \mathit{Var}(\mathbf{clock})$.

A. Syntax

A TCFA is a tuple $(\mathit{Loc}, \mathit{Urg}, \mathit{Var}, \hookrightarrow, \mathit{Inv}, \ell_0)$ where

- $(\mathit{Loc}, \mathit{Var}, \hookrightarrow, \ell_0)$ is a CFA (with $\mathbf{clock} \in \mathit{Type}$),
- $\mathit{Urg} \subseteq \mathit{Loc}$ is a set of urgent locations that model locations where time shouldn't pass, and
- $\mathit{Inv} : \mathit{Loc} \rightarrow \mathit{Form}$ is a function that maps invariants to locations.

Moreover, we assume that all atomic formulas that contain clock variables are clock constraints.[1]

As can be seen from the definition, a TCFA can either be considered a CFA extended with clock variables, urgent locations and location invariants, or alternatively, as a generalized TA where guards and clock resets are represented as statements. As a consequence, optimizations from both areas (e.g. large block encoding [14]) might be applicable.

As an example, Figure 1 depicts Fischer's protocol as a TCFA. Here, a, b and i are constant values of type $\mathbf{int}$.

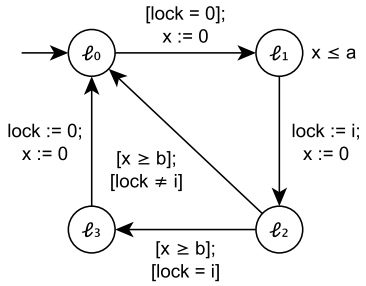


Fig. 1. Fischer's protocol as a TCFA

[1]Note that the formalism is sensible even if this assumption is not made. However, in order to apply the theory of timed automata for verification, it has to be assumed.

B. Concrete Semantics

The semantics of a TCFA is $(S, Act, \rightarrow, I)$ where

- $S = \{\ell_0\} \times \mathit{State}$,
- $I = \{(\ell_0, \mathcal{S}_0) \in S \mid \mathcal{S}_0 \models \mathit{Inv}(\ell_0)\}$,
- $Act = \mathit{Stmt} \cup \mathbb{R}_{\geq 0}$,
- and a transition $t \in \rightarrow$ of the transition relation $\rightarrow \subseteq S \times Act \times S$ is either a delay transition that increases all clocks with a value $\delta \geq 0$:

$$\frac{\ell \in \mathit{Loc} \setminus \mathit{Urg} \quad \delta \geq 0 \quad \mathcal{S}' = \mathit{Delay}(\mathcal{S}, \delta) \quad \mathcal{S}' \models \mathit{Inv}(\ell)}{(\ell, \mathcal{S}) \xrightarrow{\delta} (\ell, \mathcal{S}')}$$

or a discrete transition that models the execution of a statement $s \in \mathit{Stmt}$:

$$\frac{\ell \overset{s}{\hookrightarrow} \ell' \quad \mathcal{S}' \in \mathit{Succ}(\mathcal{S}, s) \quad \mathcal{S}' \models \mathit{Inv}(\ell')}{(\ell, \mathcal{S}) \xrightarrow{s} (\ell', \mathcal{S}')}$$

C. Abstract semantics

The abstract semantics of a TCFA can simply be defined by extending the transfer relation with transitions that abstract time delay. For TCFAs, the transfer relation is of the form $\rightsquigarrow \subseteq E \times (\mathit{Stmt} \cup \{\mathbf{delay}\}) \times E$, and the following additional property holds:

- $\bigcup_{s \in \gamma(e)} \{s' \in S \mid s \xrightarrow{\delta} s'\} \subseteq \bigcup_{e \overset{\mathbf{delay}}{\rightsquigarrow} e'} \gamma(e')$ for all $e \in E$ and $\delta \in \mathbb{R}_{\geq 0}$.

Alternatively, for the analysis of reachability properties, an abstract combined step semantics [7] can be defined where a transition is a combination of a single delay and a discrete transition.

D. Connection to TAs and CFAs

The formulation above admits a simple description of both CFAs and TAs. A timed automaton can be considered a TCFA where $\mathit{Clock} = \mathit{Var}$, $\mathit{Urg} = \emptyset$ and only statements of the form $[\varphi](\ ;\ x := 0)^*$ are allowed where $x \in \mathit{Clock}$. Here, $[\varphi]$ is a guard and $x := 0$ is a clock reset. A CFA on the other hand is a TCFA where $\mathit{Clock} = \emptyset$, $\mathit{Urg} = \mathit{Loc}$ and $\mathit{Inv}(\ell) = \mathbf{true}$ for all $\ell \in \mathit{Loc}$.

Moreover, given a TCFA with $\mathit{Inv}(\ell_0) = \mathbf{true}$, it can be transformed to a semantically equivalent CFA by applying the following simple steps:

- *Eliminating clock variables*. For all variables $x : \mathbf{clock}$ of the TCFA, the CFA has a variable $x : \mathbf{real}_{\geq 0}$ such that $\mathit{Dom}(\mathbf{real}_{\geq 0}) = \mathit{Dom}(\mathbf{clock}) = \mathbb{R}_{\geq 0}$.
- *Eliminating location invariants*. For all edges (ℓ, s, ℓ') of the TCFA, the CFA has an edge $(\ell, [\mathit{Inv}(\ell)]\ ;\ s\ ;\ [\mathit{Inv}(\ell')], \ell')$. Naturally, invariants equivalent to $\mathbf{true}$ can be omitted.
- *Simulating delay*. For all locations $\ell \in \mathit{Loc} \setminus \mathit{Urg}$ of the TCFA, the CFA has an edge $(\ell, \mathbf{delay}, \ell)$ that simulates delay steps. Here, $\mathbf{delay}$ stands for the statement

$$\mathbf{havoc}\ \delta\ ;\ x_1 := x_1 + \delta\ ;\ \ldots\ ;\ x_n := x_n + \delta\ ;\ [\mathit{Inv}(\ell)]$$

where $\delta : \mathbf{real}_{\geq 0}$ is a distinguished delay variable and $\{x_1, \ldots, x_n\} = \mathit{Clock}$.

Figure 2 shows the resulting CFA for Fischer's protocol.

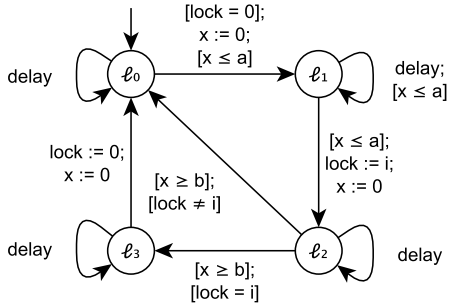


Fig. 2. Fischer's protocol as a CFA

E. Reachability Analysis of TCFAs

The main advantage of the above formulation is that it admits verifiers to be built compositionally, in the spirit of configurable program analysis [15]. More precisely, given abstractions $\mathcal{E}_{data}$ for data variables and $\mathcal{E}_{time}$ for clock variables with respective transfer relations $\rightsquigarrow_{data}$ and $\rightsquigarrow_{time}$, a simple analysis can be built that explores the two aspects independently and is a full-fledged verifier for the complete system. Here, $\mathcal{E}_{data}$ is basically a verifier for software that operates on CFAs, and $\mathcal{E}_{time}$ a verifier for timed automata.

As a simple example, Figure 3 illustrates the abstract state space of Fischer's protocol where $\mathcal{E}_{data}$ is predicate abstraction over a single predicate $\mathit{lock} = i$ (for $i \neq 0$), and $\mathcal{E}_{time}$ is zone abstraction. With both timing and data handled with an appropriate abstraction, a compact over-approximation of the concrete state space is obtained that enables sound and efficient reachability analysis of the system.

IV. Conclusions and Future Work

In this paper, we have described the formalism of timed control flow automata that is an extension of control flow automata with notions of timed automata. We have compared it to the original formalisms, and highlighted a simple method to build verifiers for the formalism by combining verifiers for CFAs and TAs.

In the future, we plan to implement such combined analyses and investigate them in depth. Moreover, to enable modeling of industrial systems, the formalism can be extended with syntax and semantics for parametric behavior and concurrency based on shared variables and handshake synchronization.

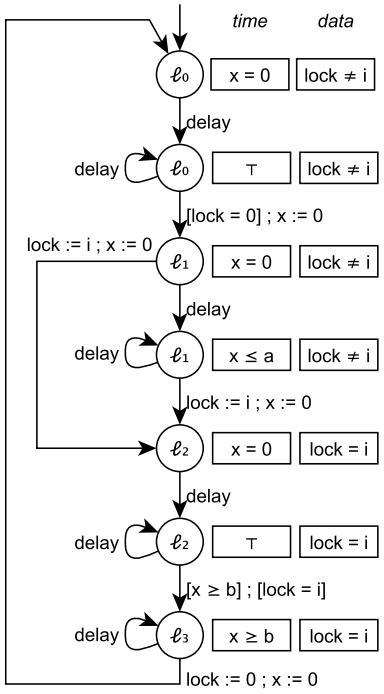


Fig. 3. Abstract Reachability Graph for Fischer's protocol

References

[1] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, no. 2, pp. 183–235, 1994.

[2] G. Behrmann, A. David, K. G. Larsen, J. Håkansson, P. Petterson, Y. Wang, and M. Hendriks, "UPPAAL 4.0," in *Third International Conference on the Quantitative Evaluation of Systems - (QEST'06)*. IEEE, 2006, pp. 125–126.

[3] J. Bengtsson, J. Bengtsson, W. Yi, and W. Yi, "Timed automata: Semantics, algorithms and tools," in *Lectures on Concurrency and Petri Nets*, 2004, vol. 3098 LNCS, pp. 87–124.

[4] G. Morbé, F. Pigorsch, and C. Scholl, "Fully Symbolic Model Checking for Timed Automata," in *Computer Aided Verification*, 2011, vol. 6806 LNCS, pp. 616–632.

[5] A. Carioni, S. Ghilardi, and S. Ranise, "MCMT in the Land of Parameterized Timed Automata," *Proceedings of VERIFY*, pp. 1–16, 2010.

[6] R. Kindermann, T. Junttila, and I. Niemelä, "Beyond Lassos: Complete SMT-Based Bounded Model Checking for Timed Automata," in *Formal Techniques for Distributed Systems*, 2012, pp. 84–100.

[7] ——, "SMT-based Induction Methods for Timed Systems," *Formal Modeling and Analysis of Timed Systems*, vol. 7595 LNCS, pp. 171–187, 2012.

[8] T. Isenberg and H. Wehrheim, "Timed Automata Verification via IC3 with Zones," in *Formal Methods and Software Engineering*, 2014, pp. 203–218.

[9] T. Isenberg, "Incremental Inductive Verification of Parameterized Timed Systems," in *Application of Concurrency to System Design (ACSD)*, 2015, pp. 1–9.

[10] K. Hoder and N. Bjørner, "Generalized Property Directed Reachability," in *Theory and Applications of Satisfiability Testing SAT 2012*, vol. 7317 LNCS, 2012, pp. 157–171.

[11] H. Hojjat, P. Rümmer, P. Subotic, and Wang Yi, "Horn Clauses for Communicating Timed Systems," *Electronic Proceedings in Theoretical Computer Science*, vol. 169, pp. 39–52, 2014.

[12] D. Beyer, "Software Verification and Verifiable Witnesses," in *Tools and Algorithms for the Construction and Analysis of Systems*, 2015, vol. 9035 LNCS, pp. 401–416.

[13] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-guided abstraction refinement for symbolic model checking," *Journal of the ACM*, vol. 50, no. 5, pp. 752–794, 2003.

[14] D. Beyer, A. Cimatti, A. Griggio, M. E. Keremoglu, S. F. Univers, and R. Sebastiani, "Software Model Checking via Large-Block Encoding," in *Formal Methods in Computer-Aided Design*, 2009, pp. 25–32.

[15] D. Beyer, T. A. Henzinger, and G. Théoduloz, "Configurable Software Verification: Concretizing the Convergence of Model Checking and Program Analysis," in *Computer Aided Verification*, 2007, vol. 4590 LNCS, pp. 504–518.

Overview and Prospects of Brain-Computer Interface Technology for Prosthetic Limbs

Kristóf Várszegi, Béla Pataki
Department of Measurement and Information Systems
Budapest University of Technology and Economics
Budapest, Hungary
Email: kristof.varszegi@sch.bme.hu, pataki@mit.bme.hu

***Abstract*— Accidents and diseases may lead to the amputation of limbs. The development of electromechanical prostheses aims to restore their function, hence granting a complete life again for injured people. But how may a prosthetic arm know what the user wants to do? Among the biggest challenges in restoring full limb function is connecting user intents to prosthesis control. This is the role of Brain-Computer Interfaces. Such devices process in real time the EEG signal acquired from the user's brain and in the case of electromechanical prostheses infer various intentions which concern the movement of the replaced limb. Intents serve as the basis of the control signals. However, extracting information from EEG recordings is not straightforward. The current state of the technology is still far from restoring full limb function. This paper provides a brief overview of current BCI technology for limb restoration and outlines the future prospects of neural controlled electromechanical prostheses.**

Index Terms—BCI; prosthesis; EEG; signal processing; motion intent

I. The EEG processing Pipeline in BCIs

From being captured to finally becoming a control signal, the EEG data goes through multiple transformations. These vary over different experiments and implementations. However, the main stages of the signal processing pipeline roughly follow a regular structure:

1. Signal acquisition
2. Artifact rejection
3. Time-domain filtering
4. Spatial filtering
5. Feature generation
6. Classification

The most crucial points of the pipeline are briefly summarized in this section.

A. Acquiring the EEG Signal

An EEG recording system mainly consists of electrodes, amplifiers with filters, analog-digital converters and a recording computer [1]. Active electrodes are placed at the parts of interest (e.g. the primary motor cortex), and the reference electrode is usually placed on the top of the head, the ear or the mastoid. The electrodes pick up the signal from the scalp, and the amplifiers magnify the microvolt magnitude signals into a range where they can be digitalized with proper accuracy. Next a computer (desktop, embedded etc.) stores and processes the obtained data according to the purpose.

The recorded EEG signal usually contains artifacts which should be removed either manually by experts or automatically. These undesired components are usually higher in amplitude and different in shape than a clean signal.

B. Defining the Expected Signal Features

For creating a processing algorithm, one must determine what signal features are expected in relation with the limb movement, both in temporal and spatial domain. There is a number of features of the EEG signal used for operating a BCI, such as Evoked Potentials (EP) [1], the Event-Related Desynchronization (ERD) and Synchronization (ERS) [2], and the Bereichtshaftspotential (or Readiness Potential) [3].

C. Time-Domain Filtering

The purpose of time-domain filters is to eliminate (to the maximal possible extent) DC and high frequency noise and power line (50/60 Hz) harmonic interference. DC and high-frequency noise can be reduced using a band pass filter, while the power line harmonic interference is usually cut out with a notch filter. Besides the elimination of the noise, the time-domain filter may also separate the EEG signal into different frequency bands (alpha, beta, gamma etc.), to help the extraction of relevant features later on. Such multi-band filters are called filter banks.

D. Spatial Filtering

The EEG is usually acquired from multiple electrodes spread over the scalp. The signals picked up by different electrodes will inevitably contain some degree of redundancy. The purpose of spatial filtering is to remove this crosstalk between the electrodes and leave only the local signal component. The most common techniques for spatial filtering are the Principal Component Analysis (PCA) [4], the Independent Component Analysis (ICA) [5], the Common Spatial Pattern (CSP) method [6]. CSP performs better than PCA or ICA, but it requires much more computation.

E. Class Feature Generation

Signals must be transformed into features computed from record portions, by which the instances belonging to different classes can be best discriminated. In case of BCIs usually EEG signal values in a 1 or 2 seconds long window are used as instances.

This work was partially supported by the ARTEMIS JU and the Hungarian National Research, Development and Innovation Fund in the frame of the R5-COP project.

A trivial shortcut at this step is simply to feed the filtered signal to the classification algorithm. Interestingly, this might be appropriate when using highly adaptive classification methods such as Artificial Neural Networks, which actually may find the best suitable transformation for the input data [7]. Features can also be the amplitudes of the oscillations within given frequency bands. A common feature generation approach is calculating the power of the signal and averaging it over a time period, for every channel. A more complex feature is the covariance matrix of the multi-channel values during an epoch. The advantage of using a covariance matrix is that it contains both per-channel and inter-channel information [8].

F. Classification

Classification is the problem of identifying the category to which the new observation belongs, on the basis of the training with a set of data instances whose membership category is known. Popular classifiers include the Linear Discriminant Analysis (LDA) [9], Logistic Regression (LR) [10], Support Vector Machine (SVM) [11], and the Artificial Neural Network [12] along with its architectural variants. Compared to LDA and LR, the SVM and the ANN are more computationally intensive, but they have higher performance.

II. Related Works

Promising results were published in [13]. An algorithm was developed that allowed a man to grasp a bottle and other objects with a prosthetic hand. The system was measuring the EEG signal and joint angular velocities. In the demonstrations, a 56-year-old man, whose right hand had been amputated, grasped objects including a water bottle and a credit card with a prosthetic hand. The subject managed to grasp the selected objects 80 percent of the time. This experiment showed that it is feasible to extract detailed information on intended grasping movements from the EEG along with joint angular velocity. It also provided evidence that the acquired signals predicted the movement, rather than reflecting it.

A rather complex method for the classification of motor imagery is described in [14]. The experiment involved two of the BCI Competition IV datasets, which contain 4 classes of motor imagery EEG trials: left hand, right hand, foot, and tongue. The processing algorithm, called Filter Bank Common Spatial Pattern (FBCSP) consists of four progressive stages of signal processing and machine learning: time-domain filter bank with Chebyshev Type II band-pass filters, CSP spatial filter, feature selection with the MIBIF (Mutual Information-based Best Individual Feature) and MIRSR (Mutual Information-based Rough Set Reduction) algorithms, then classification of the selected CSP features using the Naive Bayesian Parzen Window method. The pipeline is depicted in Fig. 1. The best results yielded a kappa value of 0.572 and 0.599 which makes this method promising for motor imagery classification.

An interesting improvement to the CSP method is described in [15]. In this study, the outputs from different CSP subspaces are combined by majority voting, as depicted in Fig. 2. The main advantage of such classifier ensemble is that a combination of similar classifiers is very likely to outperform a single classifier on its own. During the EEG recording from

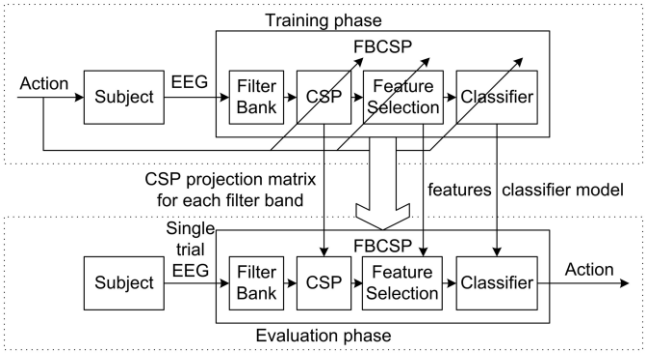


Fig. 1. Architecture of the filter bank common spatial pattern (FBCSP) algorithm for the training and evaluation phases [14].

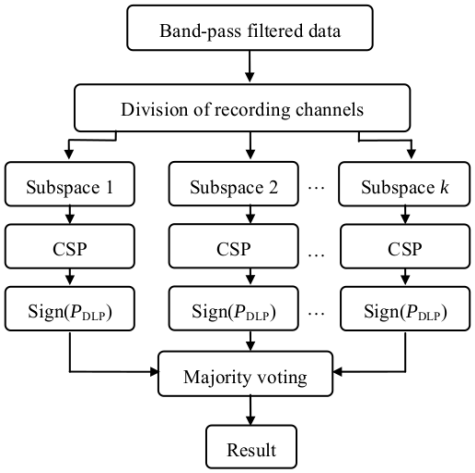


Fig. 2. The diagram of the classifier for EEG signal classification in [15].

three healthy, right-handed participants, motor imagery tasks were to be performed and the discrimination of two different movements were studied. The classification features are obtained by projecting the signal using spatial filters, and calculating the difference of log-power values coming from two tasks. The sign of the resulting feature is interpreted as the predicted class in case of a single classifier. With 10 different spatial filters 10 feature values were computed and summed to represent the majority voting method (being aware that in the case of 10 features, the equality of votes may occur). The sign of the final sum is the basis of decision between two classes. According to the study, this CSP ensemble method outperformed LDA classifiers and SVMs, making it a promising method for BCI applications.

III. The Main Issues of BCIs

Decoding the brain activity to obtain useful control signals is extremely difficult. The first issue is that EEG signals have a poor signal-to-noise ratio. One reason of this is that the electric activity is recorded on the scalp, and the skull is a massive obstacle for neural signals. There are methods to temporarily implant electrodes in the patients' brain, but usually noninvasive methods are preferred. There are several noninvasive electrode systems which reduce the scalp contact impedance by applying a saline solution to the recording

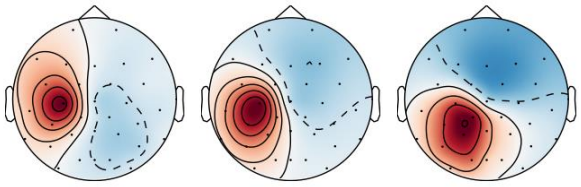


Fig. 3. Spatial patterns of the EEG from three different subjects during the execution of a specific movement. [8]

spots. Assembling an EEG recording setup can also be cumbersome, as it is difficult to place the electrodes properly and their signal quality must be checked. A setup with a saline solution is even more complicated because of the handling of the liquid.

Another issue is the accurate detection of limb motion intent from the digitalized recordings. Accurate inference also requires high spatial resolution over the scalp for the maximum possible information. Unfortunately the spatial resolution is not only a matter of electrode density. As only large populations of active neurons can generate electric potentials high enough for recording over the scalp with satisfactory signal-to-noise ratio, there is a limit on the localization of brain activity. The digital processing of the EEG with sometimes more than 100 channels involves intricate mathematics either way.

Many methods build on some form of the Bayesian inference, providing a transparent analytical transformation of the raw data into intention probabilities. Besides these, there are highly adaptive algorithms with large number of parameters. Unfortunately most of them resemble a black box, where it is difficult to see what kind of actual transformations such tools perform. However, highly adaptive methods such as various Artificial Neural Network architectures perform outstandingly in several situations, therefore their examination is also important.

The subject-specificity of the electric activity of the human brain is also an issue. An ideal, plug-and-play prosthetic device would be expected to function perfectly after being attached to the user, for which subject-invariant motion intent detection models are required in advance. This expectation seems unlikely as brain activity varies over different subjects during movement tasks, as illustrated in Fig. 3.

Knowledge about the structure and function of the human brain is essential to decide where to place recording electrodes. In case of hand motions, the most important part is the motor cortex. It is involved in the planning, control, and execution of voluntary movements. The proper selection of scalp areas may help minimizing the number of recording channels, making the digital signal processing easier.

The development of BCIs requires expertise from a multitude of areas such as neurology, electrical engineering, and mathematics, forcing experts from different fields to join in teams, which further adds to the difficulty of the problem.

IV. Concepts for Future BCI Development

A. Intracranial Recording Methods

As mentioned before, one of the major obstacles is the low signal-to-noise ratio of the noninvasive EEG as well as the limited spatial resolution. In neuroscientific research for example, intracranial recording methods are studied for cortical mapping [16]. The invasive technique of electrocorticography (ECoG) yields signals that have an exceptionally high signal-to-noise ratio, less susceptibility to artifacts than EEG, and higher spatial and temporal resolution. ECoG involves measurement of the brain actvity using electrodes that are implanted subdurally on the surface of the brain. ECoG data are often hard to obtain because of the risks and limitations of the invasive procedures involved, and the need to record within clinical settings.

It is possible that future recording setups will comprise permanently implanted intracranial electrode grids, which can be connected to an external signal processing system in a plug-in manner. Besides the medical implications of the installation of such system, exposing the brain activity raises ethical questions as well. However, these concerns may be outweighted by the advantage of possibly regaining limb functions for physically impaired people.

B. Incorporating Electromyography

Electromyography (EMG) is a technique for recording and evaluating the electrical activity of skeletal muscles [17]. EMG signals can be used for prosthesis control [18]. The activity of muscle neurons essentialy represent the patient's movement intentions, therefore recognizing patterns in an electromyogram in real-time yields a control signal for a prosthetic limb. There are numerous functioning myolelectric prostheses on the market, and such devices are under intense development. There are for example myoelectric prosthetic arms which are capable of multiple different grasps [19] which can be selected according to the user's intentions.

Myoelectric prostheses may also be limited in the resolution of movements, but the easy acquisition and processing of the EMG signals make such devices convenient. This advantage is exactly what it makes EMG signals appealing in BCI-s as well. EMG signals may provide a relevant amount of information to that obtained from EEG, leading to a possibly more accurate and delicate motion intent detection. Future BCIs may utilize electrode grids over various muscles as well as over the scalp, and record body-wide neural activity to infer movement intentions.

C. Applying Image Processing Methods to the Brain Activity Map

The EEG signals are recorded over the scalp, which can be approximated with a spherical surface, or even with a plane if few electrodes are placed closely to each other. The measured electric potential values are distributed in a 2D space as illustrated in Fig. 3. This approach makes EEG data analysis feasible for image processing. 2D filters therefore might be used to extract features from the brain activity map. Determining the coefficients of such filters can be left to highly adaptive tools such as ANNs. Convolutional Neural Networks (CNNs) provided remarkable results for example in

detecting hand motions [7],[8], while they relied on the spatial distribution only to a small extent.

The potential of CNNs in BCI applications could be deeply exploited by involving experience from the field of image processing. An initial challenge is the transformation of the scalp space into an image. The EEG signal provides spatially noncontinuous information, but a continuous scalp current density function can be obtained for example with fitting spherical splines [20]. The mapping of such spherical function to a flat image is another challenge, but seems to be analogous to the map projection problem as the preservation of geometrical distances to a desired extent is important. However, the projection problem could possibly be bypassed with CNN layouts tailored for spherical convolution.

D. Object Detection and Prosthesis Control via Visual Input

The possible sensitivity of motion intent detection using EEG and even EMG signals seems to be limited. However, the environment may also provide information about the possible user intentions. A prosthetic device could be fitted with one or more cameras throught which it could analyze the surrounding objects and infer possible actions. These could be listed on a small screen built in the device or displayed on a Head-Up Display (HUD) realized in the user's glasses. The selection from the proposed actions could be made by the user through motor imagery which would be detected from the neural activity. The prosthesis could also take care of the movement control using cameras as well as tactile sensors.

Object detection in images is an intense research topic worldwide, for example there is significant activity around the ImageNet Large Scale Visual Recognition Challenge [21], which is a competition with the goal of classifying images of a large database. An efficient algorithm could process video frames in real time, hence it could track objects within a view.

An example of this concept is the following. When a prosthetic arm detects hand motion intent from the wearer's physiological signals, and recognizes a particular door and a keyhole right in front of the user with a built-in camera, the control system could infer that the person wants to unlock the door. Then the prosthetic arm could automatically position and insert the appropriate key in the hole and open the lock, therefore assisting the wearer in entering the room or building.

V. Summary

This paper presents an overview of the EEG signal processing pipeline in BCIs and also of the main issues of current BCI technology. Four proposals were outlined for the future of BCI development for prosthetic limbs. First, the application of intracranial electrodes, targeting the issue of the signal-to-noise ratio and the spatial resolution as well. Second, the incorporation of EMG to gather more information about movement intentions. Third, the application of image processing methods to enhance motion intent detecion algorithms. Last but not least the utilization of cameras for tracking surrounding objects to support prosthetic control. Presently there is moderate hope that relying solely on EEG will lead to fully functional prosthetic limbs. However, the incorporation of additional physiological measurements and environmental data might result in significant advancement towards this goal.

References

[1] M. Teplan, "Fundamentals of EEG Measurement," Measurement Science Review, vol. 2, section 2, pp. 1-11, 2002.

[2] S. Sanei and J.A. Chambers, "EEG Signal Processing," John Wiley & Sons Ltd, 2007.

[3] H.H. Kornhuber and L. Deecke, "Hirnpotentialänderungen bei Willkürbewegungen und passiven Bewegungen des Menschen: Bereitschaftspotential und reafferente Potentiale," Pflügers Archiv, vol. 284, pp. 1–17, 1965.

[4] K. Pearson, "On Lines and Planes of Closest Fit to Systems of Points in Space," Philosophical Magazine vol. 2, pp. 559–572, 1901.

[5] A. Hyvärinen and E. Oja, "Independent Component Analysis: Algorithms and Applications," Neural Networks vol. 13, pp. 411–430, 2000.

[6] J. Müller-Gerking, G. Pfurtscheller, and H. Flyvbjerg, "Designing optimal spatial filters for single-trial EEG classification in a movement task," Clinical Neurophysiology vol. 110, pp. 787–798, 1999.

[7] blog.kaggle.com, "Grasp-and-Lift EEG Detection Winners' Interview: 3rd place, Team HEDJ," 2015. http://blog.kaggle.com/2015/10/05/grasp-and-lift-eeg-detection-winners-interview-3rd-place-team-hedj. [Accessed: 2 January 2016]

[8] A. Barachant and R. Cycon, "Code and documentation for the winning solution in the Grasp-and-Lift EEG Detection challenge," 2015. https://github.com/alexandrebarachant/Grasp-and-lift-EEG-challenge. [Accessed: 2 January 2016]

[9] K. Teknomo, "Linear Discriminant Analysis Tutorial," 2015. http://people.revoledu.com/kardi/tutorial/LDA/LDA.html. [Accessed: 2 January 2016]

[10] C.R. Shalizi, "Advanced Data Analysis from an Elementary Point of View," Carnegie Mellon University, 2012.

[11] C.J.C. Burges, "A Tutorial on Support Vector Machines for Pattern Recognition," Data Mining and Knowledge Discovery vol. 2, pp. 121–167, 1998.

[12] M.A. Nielsen, "Neural Networks and Deep Learning," Determination Press, 2015.

[13] H.A. Agashe, A.Y. Paek, Y. Zhang, and J.L. Contreras-Vidal, "Global cortical activity predicts shape of hand during grasping," Frontiers in Neuroscience, 9.121., 2015.

[14] K.K. Ang, Z.Y. Chin, C. Wang, C. Guan, and H.A. Zhang, "Filter bank common spatial pattern algorithm on BCI competition IV Datasets 2a and 2b," Frontiers in Neuroscience, 6.39., 2012.

[15] X. Lei, P. Yang, P. Xu, T.-J. Liu, and D.-Z. Yao, "Common Spatial Pattern Ensemble Classifier and Its Application in Brain-Computer Interface," Journal of Electronic Science and Technology of China, 7.1., 2009.

[16] N.J. Hill et al., "Recording Human Electrocorticographic (ECoG) Signals for Neuroscientific Research and Real-time Functional Cortical Mapping," J. Vis. Exp., vol. 64, e3993, 2012.

[17] H. Turker, "Electrodiagnosis in New Frontiers of Clinical Research," InTech, 2013.

[18] S. Sudarsana, E.C. Sekaran, "Design and Development of EMG Controlled Prosthetics Limb," Procedia Engineering, vol. 38., pp. 3547-3551, 2012.

[19] TED Talks, "Todd Kuiken: A prosthetic arm that "feels"," 2011. https://www.ted.com/talks/todd_kuiken_a_prosthetic_arm_that_feels. [Accessed: 2 January 2016]

[20] F. Perrin, J. Pernier, O. Bertrand, and J.F. Echallier, "Spherical splines for scalp potential and current density mapping," Electroencephalography and Clinical Neurophysiology, vol. 72., issue 2., pp. 184-187., February 1989.

[21] O. Russakovsky et al., "ImageNet Large Scale Visual Recognition Challenge," International Journal of Computer Vision (IJCV), vol 115., pp. 211-252, 2015.