FMIS 2019

The 8th Formal Methods for Interactive Systems workshop

Participants proceedings

Porto, October 7, 2019

Preface

This volume contains the papers presented at FMIS 2019: **The 8th Formal Methods for Interactive Systems** workshop held on October 7, 2019 in Porto. The aim of the workshop was to bring together researchers from a range of disciplines within computer science (including HCI) and other behavioural disciplines, from both academia and industry, who are interested in both formal methods and interactive system design.

The goal of the FMIS workshop series is to grow and sustain a network of researchers interested in the development and application of formal methods and related verification and analysis tools to HCI and usability aspects of ubiquitous systems. Reducing the risk of human error in the use of interactive systems is increasingly recognised as a key objective in contexts where safety, security, financial or similar considerations are important. These risks are of particular concern where users are presented with novel interactive experiences through the use of ubiquitous mobile devices in complex smart environments. Formal methods are required to analyse these interactive situations. In such complex systems analysis and justification that risk is reduced may depend on both qualitative and quantitative models of the system.

This volume was produced using EasyChair.

September 19, 2019

José Creissac Campos
Steve Reeves
(FMIS 2019 co-chairs)

Programme Committee

Table of Contents

Examples of the application of formal methods to interactive systems: abstract

Michael D. Harrison[1][0000−0002−5567−9650]

School of Computing, Newcastle University
Newcastle upon Tyne, UK
michael.harrison@ncl.ac.uk
http://www.ncl.ac.uk/computing/people/profile/michael.harrison

Abstract. Formal methods in interactive systems can be used to analyse how systems support use with a clarity that is not possible with more traditional development approaches. However, the processes involved are complicated and do not fit well with those whose primary concern is user interfaces. The full paper reflects on the tools that are used and the problems that hinder their accessibility and comments on tool developments that could lead to wider use of these techniques. The role that existing methods and tools can play in analysing interactive systems will be explored through concrete examples involving the use of the PVS theorem proving assistant and the IVY toolset. Examples will focus on:

– the formulation and validation of models of interactive systems;
– the expression of use related requirements, particularly in the context of usability engineering and safety analysis;
– the generation of proofs that requirements hold true and making sense when proof fails.

Examples will be taken from existing standalone medical devices including examples from part of a safety analysis of a device leading to product.

Keywords: Formal verification · Automated reasoning tools · Interactive computing systems.

1 Summary

The use of formal methods can provide benefits in the development and analysis of interactive systems. However many of these benefits are potential rather than actual because of the many obstacles to their use. This is of particular significance in the context of interactive systems because the developers and analysts of such systems may not be computer scientists. Their focus and expertise may be the domain in which the system is to be designed, or the role of the user and how the user interface supports that role. Development teams can be small. This is particularly so in the development of medical devices. The full paper explores two specific tools that support formal methods using examples of interactive systems, namely PVS [6] (a theorem proving assistant) and IVY [1] (a tool that interfaces with the SMV [2] model checker).

The exploration considers a small set of examples, all of which concern medical devices. Medical devices are of particular interest for two reasons. Firstly, they are often safety critical and use related errors are a widespread problem for the community. Secondly the teams involved in their development are often small. It is typical that a new medical device is built by a research team who are focussed on the science associated with the device rather than its usability. In the paper we explore through the examples the following issues.

– the formulation and validation of the models that are intended to capture the key use characteristics of these devices;
– the expression of use related requirements: these requirements could be for example design heuristics or safety requirements derived from a risk log;
– the process of proving and demonstrating that a requirement holds true of the model and, by extension, is true of the existing or intended device.

These issues will be explored through three examples: the safety analysis of a neonatal dialysis machine [3]; the user centred design of a pill dispenser [4] and an infusion pump [5]. It is not intended that the paper is exhaustive in its consideration of these issues. Instead practical examples of the use of formal techniques are used to illustrate them in the development and analysis of interactive systems. The full paper will be concluded by briefly considering extensions to tools that would aid the practical use of formal tools in the context of these examples.

References

1. Campos, J.C., Sousa, M., Alves, M.C.B., Harrison, M.D.: Formal verification of a space system's user interface with the IVY workbench. IEEE Transactions of Human Machine Systems **46**(2), 303–316 (2016)
2. Cimatti, A., Roveri, M., Olivetti, E., Keighren, G., Pistore, M., Roveri, M., Semprini, S., Tchaltsev, A.: NuSMV 2.3 user manual. Tech. rep., ITC-IRST, Trento, Italy (2007), nusmv.irst.itc.it/NuSMV/tutorial/v23/tutorial.pdf
3. Harrison, M.D., Freitas, L., Drinnan, M., Campos, J.C., Masci, P., di Maria, C., Whitaker, M.: Formal techniques in the safety analysis of software components of a new dialysis machine. Science of Computer Programming **175**, 17 – 34 (2019). https://doi.org/https://doi.org/10.1016/j.scico.2019.02.003, http://www.sciencedirect.com/science/article/pii/S0167642318300819
4. Harrison, M., Masci, P., Campos, J.: Formal modelling as a component of user interface design. In: Mazzara, M., Ober, I., Salaün, G. (eds.) Software Technologies: Applications and Foundations STAF 2018 collocated workshops (revised selected papers). pp. 274–294. No. 11176 in Lecture Notes in Computer Science, Springer-Verlag (2018)
5. Harrison, M., Masci, P., Campos, J.: Verification templates for the analysis of user interface software design. IEEE Transactions on Software Engineering **45**(8), 802–822 (2019)
6. Owre, S., Rushby, J., Shankar, N.: PVS: A prototype verification system. In: Kapur, D. (ed.) Eleventh International Conference on Automated Deduction (CADE). Lecture Notes in Artificial Intelligence, vol. 607, pp. 748–752. Springer-Verlag (1992)

A Survey on Formal Methods for Interactive Systems

Pascal Béger, Sebastien Leriche, and Daniel Prun

ENAC, Université de Toulouse - France
{pascal.beger,sebastien.leriche,daniel.prun}@enac.fr
http://lii.enac.fr/

Abstract. Our research team is specialized in human-computer systems and their engineering, with focus on interactive software systems for aeronautics (from cockpits to control towers). This context stands out by the need for certification, such as DO-178 or ED-12. Today, formal methods are pushed forward, as one of the best tools to achieve the verification and validation of properties, leading to the certification of these systems.
Interactive systems are reactive computer systems that process information from their environment and produce a representation of their internal state. They offer new rich interfaces with sophisticated interactions. Their certification is a challenge, because the validation is often a human based process since traditional formal tools are not always suitable to the verification of graphical properties in particular.
In this paper, we explore the scientific work that has been done in formal methods for interactive systems over the last decade, in a systematic study of publications in the International Workshop on Formal Methods for Interactive Systems. We describe an analytical framework that we apply to classify the studied work into classes of properties and used formalisms. We then discuss the emerging findings, mainly the lack of papers addressing the formal specification or validation of perceptibility properties. We conclude with an overview of our future work in this area.

Keywords: interactive software · formal methods · verification · graphical properties.

1 Introduction

1.1 Aim and scope of the article

Interactive systems are reactive computer systems that process information (mouse clicks, data entries, etc.) from their environment (other systems or human users) and produce a representation (sound notification, visual display, etc.) of their internal state [13, 59]. They now have an increasingly important place among modern systems in various sectors such as aeronautics, space, medical or mobile applications. These systems offer new rich human machine interfaces with sophisticated interactions.

The preferred method for the verification and validation (V&V) of properties on interactive systems remains largely based on successive testing sessions of prototypes, performed through various experimentations involving representative end-users. For a long time, formal methods have not been very used to the verification of interactive properties. Indeed, historically, formal methods have been developed for distributed and embedded systems. The first properties studied for software and computer systems concerned safety (e.g. absence of unwanted events, boundedness) as well as program liveness (e.g. return to a given state, deadlock freedom) [63]. The main methods used to verify and validate properties of systems are model verification by model checking [25], mathematical proof [18], static analysis [43] and test processes driven by a formal model of the system under tests.

However, more and more work is being done on the development of formal methods to interactive systems. The objective is to study how these methods can be adapted to the modelling and the verification of properties involving human related characteristics. In particular, in the scope of critical domains such as aeronautics, recent updates of standards used for certification strongly recommend to use formal methods for the verification and validation of requirements of new software for aircraft cockpits ([71, 72]).

In this context, the objective of this survey is to study research activities that have been done in formal methods for the modeling, verification and validation of interactive systems, over the last decade. The aim is to draw a faithful picture of formalisms that are used to model interactive systems, set of properties that are verified and formal methods applied. From this picture, the objective is to identify strengths and weaknesses of formal approaches for interactive systems and to identify ways of improvements. More precisely, the survey highlights several points: What interactive related properties are studied? Which ones are more covered and which ones are least addressed? Are there formalisms that are widely used to model systems and study their properties? Are there any new formalisms that have emerged? Are they used on industrial critical systems or only on small academic case studies?

1.2 Methodology

Through this survey we explore the scientific work that has been done in formal methods for interactive systems over the last decade. For this purpose, we perform a systematic study of publications from a specific workshop, the International Workshop on Formal Methods for Interactive Systems (FMIS). We have selected this workshop because it covers exactly our problematic: the articles from this workshop address issues of how formal methods can be applied to interactive system design and verification and validation of their related properties. The workshop also focuses on general design and verification methodologies, and takes models and human behavior under consideration. Moreover, FMIS has reached a critical mass that makes the analysis more significative and reliable. It has taken place seven times from 2006 to 2018. Our study is based on an exhaustive review of the literature from FMIS representing 43 articles.

As we focus on the formal study of properties related to the graphical scene of interactive systems, this survey is based on a table of our choice that classifies the work that has been done about formalisation and verification of properties for interactive systems.

1.3 Plan of the article

Before reviewing the work from FMIS, we present our analytical framework (2). It is composed by definitions of properties we have sorted in different classes. We also set up a nomenclature of formalisms that have been used for the studies of the properties. From this basis, we propose an analytical grid that allows us to synthesize the review. Then the 43 articles from all the FMIS workshops are presented and analysed (3), analysis mainly directed by the studied properties and the ways of studying them.

The section 4 provides a synthesis of the review and highlights the issues in the research of formal methods for interactive systems. Finally, the section 5 concludes the discussion and presents ongoing work related to the previously highlighted issues.

2 Analytical framework

The purpose of this section is to define a framework for the analysis of the properties that have been studied for interactive systems. In order to do that, three basic questions must be considered.

- *"What properties are studied?"* This question concerns the nature of properties that have been studied and is the center of our work to determine if some properties have not been studied.

– *"What formalism is being used?"* This question allows us to show what formalisms can be used to study the properties.
– *"What is the case study?"* This question concerns the system used as the case study to illustrate the use of formal methods and its particularities.

We focus on these questions in order to highlight the range of interactive systems properties covered. It provides the means used to cover these properties. Through this survey, we explore these questions by sorting the articles by the properties studied and the means used to study those. We also provide the case study used to illustrate the studies.

2.1 Properties

As stated in the analytical framework, we firstly drive our analysis according to the studied properties. This paper organises interactive systems properties in four classes of our choice: user behavior [2], cognitive principles [29], human-machine interfaces [13], security [70]. We detail these classes below.

Several articles do not directly address interactive properties and so cannot be classified in one of these 4 classes. For these specific papers, we have defined two additional categories:

– **specification/formal definition**: gather papers dealing with the formal modeling of a system, and possibly addressing properties related to the model itself, and not centered on the interaction.
– **testing**: gather papers related to the modeling of interactive systems with the objective to generate test cases from the study of the model.

User behavior This user behavior class considers the properties related to a human user. The properties from this class are about user's actions, user's expectations about the system, user's objectives and restrictions.

– A **user goal** is a list of sub objectives that a user has to perform to achieve a greater objective related to the purposes of the system used. This goal can consist on a single task or an overall use case.
"Insert the card", "authenticate" and "choose the amount of money" are subgoals of "withdraw money".
– **User privileges** are a way to prevent a user with an unauthorized level of accreditation to perform goals the user should not.
Example: It is only possible to access our e-mails if we are connected to our e-mail system.
– The **user interpretation** can be seen as the set of assumptions of the user about the system. It can lead users to adapt their behavior in accordance with these assumptions.
For example, we are used to the shortcut Ctrl+C in order to copy some text. A novice user of a terminal could use it to copy text and close the running application because the functionality is not the same.
– The **user attention** is defined as the ability of the user to focus on a specific activity without being disturbed by irrelevant informations.
This can be seen when driving a car, the driver is focused on traffic signs, on road traffic, etc.
– The **user experience** concerns the knowledge of the user about the system. This knowledge can come from a previous use of the system or a study of the system before using it. This experience can have an impact on user interpretation.
The example given in user interpretation also illustrates the user experience: an experienced user of a terminal would not make mistakes with the Ctrl+C functionality.

Cognitive principles This cognitive principles class considers the properties related to cognitive sciences. The properties from this class are about the human user cognitive salience and load.

– The cognitive **load** is related to the task performed by the user and more specifically to its complexity. It is possible to define two types of cognitive load: intrinsic (complexity of the task) and extraneous (complexity due to the context and distractors).
  For example, a user may lose attention while interacting with too rich a graphical scene.
– The cognitive **salience** represents a user's adherence to an idea. While performing an action, it depends on the action sensory salience, its procedural cueing and the cognitive load related to the task.
  A user will be more focused on an action more in line with his convictions.

Human-machine interfaces In the human-machine interfaces properties class we consider the new properties that have arrived with these new systems. These properties are mainly specific to the problems induced by the display such as verifying the right display of informations or being aware of the latency that can appear between user actions and the display of informations.

– The **latency** is a well-known issue in rich interfaces. It concerns the delay between interactions with an application and the return of informations from it.
  If a computer processes several actions at the same time, it will take a few seconds to start a web browser.
– the **consistency** represents a system constant behavior whether for a display or a functionality regardless the current mode of the system.
  It can be seen as the use of same terminology for functions ("Exit" or "Quit" in order to define a function "close a window").
– The **predictability** is the user's ability to predict the future behavior of the system from its actual state and the way the user will interact with it.
  When closing a word processor with an unsaved document, a user knows that a pop-up will show to ask what to do between saving the document, canceling the closing or closing without saving.
– ISO 9241-11 [79] defines the **usability** as "the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use."
  It is possible to improve the usability of an "accept/decline" window by adding symbols related to the two notions such as ✓for accept and $\times$ for decline.
– The visual **perceptibility** is based on different properties such as the superposition of components, the distinction of shapes and colours.
  For example, even if a red text is above a red shape, the text will not be perceptible.

Security This security class considers the properties related to computer security such as the prevention of threats and the link between the user behavior and the possible threats.

– The **integrity** property states that, for a system that may be exposed to threats, hypothesis of the user about the application are correct and the reverse is also true.
  When we log in interfaces with two text fields, if the fields username and password are not in the expected locations, we could type the password in the clear field.
– The **threats** property focuses in defining the differents threats that may be a risk for the system.
  We can note, for example, data leaking or data manipulation.

2.2 Nomenclature of formalisms

This section will introduce formalisms and formal methods that have been used by FMIS authors in order to formalize and apply verification techniques on the properties defined in the last section. We will define the basic semantic and the properties inherent to these formalisms.

Process algebra Baeten [7] gives the history and the definition of process algebra. The author also gives examples of some formalisms from process algebra such as Calculus of Communicating Systems (CCS) or Communicating Sequential Processes (CSP). We can resume from this paper that process algebra is a set of algebraic means used to study and define the parallel systems behavior.

Authors from the FMIS workshop used formalisms from process algebra such as the CWB-NC [34] syntax for the Hoare's CSP notation [48], Language Of Temporal Ordering Specification (LOTOS) [51], probabilistic π-calculus [61], applied π-calculus [69], Performance Evaluation Process Algebra (PEPA) [47].

Specification language A specification language is a formal language that can be used to make formal descriptions of systems. It allows a user to analyze a system or its requirements and thus to improve its design.

Authors from the FMIS workshop used specification languages such as SAL [58], Z [78], μCharts [41], Spec# [11], Promela [49], PVS [73], Higher-Order Processes Specification (HOPS) [36].

Refinement A program refinement consists in the concretisation of a more abstract description of a system. The aim of this method is to verify properties in an abstract level of the description then to concretise this level while conserving the verified properties. These steps have to be done until the concrete description of the system is obtained.

Authors from the FMIS workshop used refinements processes with models such as B-method [3] or with specification languages such as Z and μCharts.

Transition systems Transition systems [8] consist in directed graphs composed of states, represented by nodes, and transitions, represented by edges. A state represents an instant in the system behavior or for a program the current value of all the variables and the current state of the program. Crossing a transition involves a change of state.

Authors from the FMIS workshop used transition systems formalisms such as UPPAAL [15], Petri nets (PN) [35], ICO models [60], finite state automata (FSA), Input/Output labeled transition system (IOLTS).

Temporal logic Properties to be verified are often expressed in the form of temporal logic formulas [42]. These formulas are based on Boolean combiners, time combiners and for some logics on path quantifiers.

Authors from the FMIS workshop used temporal logics such as computation tree logic (CTL) and linear temporal logic (LTL) [25].

3 Review

Here, we review the state of the art of formal methods applied to interactive systems. We consider research work that have been presented in the International Workshop on Formal Methods for Interactive Systems.

Our aim is to present the properties that have been studied with formal methods. From this and the questions that we asked in the section 2, we base our analysis on the grid presented in the table 6.

This grid highlights the coverage of properties depending on the formalisms. The categories formal definition/specification and testing are not interactive systems properties. However, we want to present how articles address those with formal methods. This explains the fact that there is a double vertical line in the grid. Our work addresses the visual perceptibility property from the HMI class. We highlight this by setting the perceptibility in italic beside the HMI class, separated by a dashed line.

3.1 User behavior

The table 1 summarizes the studies of the user behavior class of properties. It sorts the papers according to the properties studied (goals, privileges, interpretation, attention, emotion and experience) and formalisms used.

Table 1: Study of the **user behavior** class of properties in the FMIS workshops

	Goals	Privileges	Interpretation	Attention	Emotion	Experience
(PA) CWB-NC	[32]	[32]				
(PA) CSP	[30]			[30]		[30]
(PA) LOTOS				[80]		
(PA) PEPA						[33]
(SL) SAL	[66] [65]		[66] [67]			
(SL) HOPS	[37]					
(other) HTDL				[31]		[31]
ad-hoc formalism					[19]	

User goals Cerone and Elbegbayan [32] define user goals in the use of a web-based interface that features a discussion forum and a member list. Those are defined with the CWB-NC syntax for CSP from process algebra. These definitions allow authors to model more precisely the attended and unattended use cases.

Rukšėnas et al. [66] address the use of an authentification interface with two textboxes (user name and password). They define user goals with the specification language SAL through the definition of a cognitive architecture of user behavior. It allows authors to define the actions a user can do. Rukšėnas et al. [65] further explore the notion of user goals through their cognitive architecture.

Cerone [30] bases his work on the study of two use cases: a driving user and a user interacting with an ATM. He models the user goals with the Hoare's notation for describing CSP (process algebra). It allows him to study cognitive activities such as closure.

Dittmar and Schachtschneider [37] use HOPS (specification language) models to define user tasks and actions while solving a puzzle.

User privileges Cerone and Elbegbayan [32] define user privileges with the CWB-NC syntax for CSP. Thus, authors can model wich actions logged or non-logged users are allowed to do. This allows authors to constrain the user behavior by adding new properties in the web interface model.

User interpretation Rukšėnas et al. [66] address the user interpretation of an authentification interface. They define it with SAL through the definition of a cognitive architecture of user behavior. It allows authors to highlight the risk for the user of misunderstanding the interface depending on the display of the two textboxes. Rukšėnas and Curzon [67] study the plausible behavior of users interacting with number entry on infusion pumps. They assume that users have their own beliefs about the incremental values. They separately model the users behavior depending on their interpretation and the constraint on cognitive mismatches with LTL and the SAL model checker.

User attention and user experience Su et al. [80] study the temporal attentional limitation in the presence of stimuli on stimulus rich reactive interfaces. The cognitive model of human operators is defined with LOTOS (process algebra). The model of SRRI is based on studies of an AB task [39]. This work presents simulation results focusing on the performance of the interface in user attention.

Cerone [30] addresses user's expectations, which relies on user attention and user experience. He studies cognitive activities such as closure, contention scheduling and attention activation. He models those with the Hoare's notation for describing CSP (process algebra).

Cerone and Zhao [33] use the process algebra PEPA to model a three-way junction with no traffic lights and a traffic situation. They study the user experience in driving in such junctions. They use the PEPA Eclipse plug-in to analyse the model and determine for example the probability of possible collision.

Cerone [31] proposes a cognitive architecture for the modelling of human behavior. This work presents the Human Task Description Language (HTDL). He uses it to model properties related to user behavior such as the automatic (everyday tasks) and deliberate (driven by a goal) control and the human learning, attention and experience.

User emotion Bonnefon et al. [19] use their logical framework, an ad-hoc formalism, to model several emotions and the notion of trust. Among the emotions there is joy/distress, hope/fear, satisfaction/disappointment and fear-confirmed/relief. They also model the relation between trust and emotions.

3.2 Cognitive principles

The table 2 summarizes the studies of the cognitive principles class of properties. It sorts the papers according to the properties studied (salience and load) and formalisms used.

Table 2: Study of the **cognitive principles** class of properties in the FMIS workshops

	Salience	Load
(SL) SAL	[65] [50]	[65] [50]
(other) GUM	[50]	[50]

Rukšėnas et al. [65] define two cognitive principles, salience and cognitive load. They add those to their SAL cognitive architecture. The authors also define the link between these two principles. They illustrate these principles through the case study of a Fire Engine Dispatch Task.

Huang et al. [50] try to see if their Generic User Model (GUM) can encapsulate all the cognitive principles presented in the Doughnut Machine Experiment [4].

3.3 Human machine interfaces

The table 3 summarizes the studies of the HMI class of properties. It sorts the papers according to the properties studied (consistency, predictability, latency and usability) and formalisms used.

Table 3: Study of the **HMI** class of properties in the FMIS workshops

	Consistency	Predictability	Latency	Usability	Perceptibility
(SL) SAL		[56]		[66] [65]	
(SL) PVS	[45] [46]				
(SL/Re) μCharts				[22]	
(SL/Re) Z	[21]				
(TS) IOLTS	[14]				
(TL) LTL	[14]			[65]	
(TL) CTL	[45] [27]				
(other) Tree based WCET			[54]		

Consistency Bowen and Reeves [21] use their presentation models and refinement processes with Z to check the equivalence and the consistency between two UI designs. The presentation models allow them to ensure that controls with the same function have the same name and conversely.

Beckert and Beuster [14] provide an IOLTS model of a text-based application to guarantee consistency constraints. Their first model does not satisfy consistency constraints. They refine this model in order to satisfy the consistency constraints.

Campos and Harrison [27] provide consistency a formal definition of consistency of the Alaris GP Volumetric Pump interface in CTL. The global consistency includes: the role and visibility of modes, the relation between naming and purpose of functions, consistency of behavior of the data entry keys. They also present a part of a MAL specification of the Alarais GP infusion pump.

Harrison et al. [45] explore the consistency in the use of the soft function keys of infusion pumps through the use of MAL models translated into PVS. They define consistency properties with CTL and translate those into PVS theorems.

Harrison et al. [46] create a model of a pill dispenser from a specification in PVS. They use this specification with the PVSio-web tool to study the consistency of possible actions.

Predictability Masci et al. [56] analyse the predictability of the number entry system of Alaris GP and B-Braun Infusomat Space infusion pumps. They use SAL specifications to specify the predictability of the B-Braun number entry system.

Latency Leriche et al. [54] explore the possibility of using Worst-Case Execution-Time [64] based on trees to study the latency for interactive systems. They also present some works that have been done with graphs of activation to model interactive systems.

Usability Rukšėnas et al. [66] use their user behavior model in SAL to check usability properties of an authentification interface. They check that the property "the user eventually achieves the perceived goal" is satisfied. Rukšėnas et al. [65] further explore the use of their user model with SAL and LTL properties. They check that the property "the user eventually achieves the main goal" is satisfied in the Fire Engine Dispatch Task.

Bowen and Reeves [22] present a way of applying the specification language μCharts and efinement processes to UI designs. They use presentation models to compare two UI designs and if these UI maintain usability. They also informally describe the refinement process related to UI design.

3.4 Security

The table 4 summarizes the studies of security class of properties. It sorts the papers according to the properties studied (integrity, usability errors and threats) and formalisms used.

Table 4: Study of the **security** class of properties in the FMIS workshops

	Integrity	Usability errors	Threats
(SL) SAL		[66]	
(TS) IOLTS	[14]		
(TL) LTL	[14]		
(other) BDMP			[52]
others/ad-hoc	[6]	[6]	

Rukšėnas et al. [66] check the risk of security breach in the authentification interface with SAL properties. This highlights the fact that user interpretation can impact the security by entering the password in the wrong textbox for example.

Beckert and Beuster [14] produce a generic IOLTS (transition system) model of a text-based application. They use LTL to describe the properties of components and interpret them with IOLTS. The model is refined to guarantee integrity and to consider the problem of multi-input (if the user enters again a data if the system has not yet processed the last one) risking security breaches.

Arapinis et al. [6] present security properties related to the use of the MATCH (Mobilising Advanced Technology for Care at Home) food delivery system. They define these properties by using different formalisms such as the access control language RW and temporal logic (LTL, TCTL, PCTL).

Johnson [52] studies security properties in terms of threats that may occur on Global Navigation Satellite Systems (GNSS). He models GNSS with Boolean Driven Markov Processes (BDMP) and integrate security threats to the model.

3.5 Specification/formal definition and testing

The table 5 summarizes the studies of the specification/formal definition and testing classes. It sorts the papers according to the case (specification/formal definition and testing) and formalisms used. This section allows us to present different systems used as case studies.

The references concern the articles that address the formal definition or specification of systems. These articles do not cover the properties previously presented. We only present in this section these articles.

Specification/formal definition We sort the articles only focused in specification/formal definition by formalism used.

Process algebra Barbosa et al. [10] represent an air traffic control system with a control tower and three aircrafts as CNUCE interactors. They use ad-hoc formalism, a generic approach to process algebra, to define this representation.

Table 5: Study of the **specification/formal definition** and **testing** classes in the FMIS workshops

	Formal definition	Testing
(PA) CSP	[30]	
(PA) LOTOS	**[10]**	
(PA) π-calculus	[6]	
(PA) Prob. π-calc.	**[5]**	
(PA) PEPA	[33]	
(PA) TCBS'	**[16]**	
(SL) SAL	[56] **[12]**	
(SL) Spec#		[74]
(SL) PVS	**[55]** [45] **[62]** [46]	
(SL) Promela	**[26]**	
(SL) HOPS	[37]	
(SL/Re) μCharts	[22]	
(SL/Re) Z	[21] [23]	[23]
(Re) B/event-B	**[28]** **[68]** **[40]**	
(TS) FSM	**[81]**	
(TS) UPPAAL	**[44]**	
(TS) Colored PN	**[75]**	
(TS) GTS	**[83]**	
(TS) FSA	**[82]**	
(TS) Event act. graph	[54]	
(TS) ICO	**[75]**	
(TL) LTL	[6] **[26]**	
(TL) CTL	[45]	
(other) SAT	**[26]**	
(other) Mark. proc.	**[5]**	
(other) MAL	[27] [45]	
(other) GUM	[50]	
(other) BDMP	[52]	
(tool) Spec explorer		[74]
(tool) FEST		[74]
(tool) SMT solver		[23]
(tool) PVSio web	**[62]** [46]	
others / ad-hoc	[76] [17] [38] [6] [30] **[20]** **[81]** **[9]**	

Anderson and Ciobanu [5] builds a Markov Decision Process abstraction of a program specification expressed with a probabilistic process algebra (using π-calculus). The abstraction is then used to check the structure of specification, analyze the long-term stability of the system, and provide guidance to improve the specifications if they are found to be unstable.

Bhandal et al. [16] present the language TCBS', strongly based on the Timed Calculus of Broadcasting Systems (TCBS). They give a formal model of a coordination model, the Comhordú system, in this language.

Specification language Calder et al. [26] study the MATCH Activity Monitor (MAM), an event driven rule-based pervasive system. They model separately the system behavior and its configuration (rule set) with Promela. They derive Promela rules in LTL properties to check redundant rule with the model checker SPIN.

Bowen and Hinze [20] present early stages work using presentation models to design a tourist information system. This system displays a map on a mobile support (smartphone).

Bass et al. [12] specify in SAL the three subsystems of the A320 Speed Protection: automation, airplane and pilots. This interactive hybrid system has the potential to provide an automation surprise to a user.

Masci et al. [55] specify the DiCoT's information flow model by using PVS. They use three modelling concepts (system state, activities, task) for this specification. The authors use the example of the London Ambulance Service to illustrate their work.

Refinement Cansell et al. [28] specify an interface of e-voting corresponding to the Single Transferable Vote model without the counting algorithm. This is done by using the B method and a refinement process.

Rukšėnas et al. [68] study the global requirements related to data entry interfaces of infusion pumps. They use Event-B specifications and refinement processes with the Rodin platform to specify these requirements. These refinement processes allow the authors to check if the Alaris GB infusion pump number entry specification validate the global requirements.

Geniet and Singh [40] study an HMI composed by graphical components in form of widgets. They use the Event-B modelling language and refinement processes to model the system and analyse its behavior.

Transition system Harrison et al. [44] model the GAUDI system [53] with UPPAAL. Through the UPPAAL model, the authors can explore use cases scenario and check reachability properties for example.

Westergaard [83] uses game transition systems to define visualisations of the behavior of formal models. The example of an interoperability protocol for mobile ad-hoc networks to highlight the use of visualisations.

Thimbleby and Gimblett [81] model the interactions possibilities with key data entry of infusion pumps. They use FSM and specify those with regular expressions to model the interactions.

Silva et al. [75] formally define a system and its WIMP and Post-WIMP interactions with ICO models and colored Petri nets. These models allow them to analyse the properties inherent to the formalisms: place transitions invariants, liveness and fairness, and reachability.

Turner et al. [82] generate presentation models describing tasks and widgets based interactions sequences of an infusion pump. It is composed by five buttons (Up, Down, YesStart, NoStop, OnOff) and a display allowing interactions with the user. They use FSA to model these sequences.

Others Bhattacharya et al. [17] model soft keyboards (on-screen keyboards) with scanning and use the Fitts-Digraph model [77] to evaluate the performance of their model and the system.

Sinnig et al. [76] describe a new formalism based on sets of partially ordered sets. They use it to formally define use cases and task models.

Dix et al. [38] use an ad-hoc formalism to model physical devices (switches, electric kettle, etc.) logical states and their digital effects in another model.

Oladimeji et al. [62] present PVSio-web, a tool which extends the PVSio component of PVS with a graphical environment. They demonstrate its use by prototyping the data entry system of infusion pumps.

Banach et al. [9] consider using an Event-B model in conjunction with an SMT solver in order to proof some invariants on a hardware based components, dedicated to the acquisition and fusion of inputs from various sensors to a visually impaired and blind person's white cane (INSPEX project).

Testing Silva et al. [74] highlight a way of testing model-based graphical user interfaces. The testing process presented is as follows: a FSM model called Presentation Task Sets (PTS) is generated from a task model (CTT) with the TERESA tool [57], a Spec# oracle is generated from the FSM model with their Task to Oracle Mapping (TOM) tool, then a testing framework is used to test the system against the oracle.

Bowen and Reeves [23] use the specification language Z for specifying a calendar application. They explore the way to apply testing processes on this application. They use their presentation and interaction models to derive tests such as ensuring that the relevant widgets exist in the appropriate states and ensuring that the widgets have the required behaviors.

4 Findings

Through this survey, we have explored the study of interactive systems with formal methods. Several classes of properties have been studied and cover different aspects of interactions.

The table 6 summarizes the studies of the articles from the International Workshop on Formal Methods for Interactive Systems that has taken place seven times from 2006 to 2018. It gives a distribution of the articles in our analytical grid. We note: ✓: 1-5 articles; ✓✓: 6-10 articles; ✓✓✓: 10+ articles.

Table 6: Study of interactive systems properties in the FMIS workshops

	User behavior	Cognitive pr.	HMI Perceptibility	HMI Others	Security	Formal def.	Testing
Process algebra	✓					✓✓	
Spec. language	✓	✓		✓✓	✓	✓✓✓	✓
Refinement				✓		✓	✓
Transition systems	✓			✓	✓	✓✓	
Temporal logic		✓		✓	✓	✓	
Other / ad-hoc	✓	✓		✓	✓	✓✓	✓

High proportion of works on formal definitions and specifications We highlight the high proportion of articles that address the formal definition and the specification of interactive systems (classified in "Formal def." column of table 6). Among the 43 articles from the FMIS workshops, 34 are related to this aspect (representing approximately 80%). More than the half of those specifically address the formal definition of properties inherent to the formalisms used (invariant for B, reachability for transition systems, etc.).

Perceptibility unstudied We can note that even if several properties related to HMI have been studied, no paper addresses perceptibility properties (cf. "Perceptibility" column). In the FMIS workshops, we have not spotted studies addressing visual, sound or haptic based interactions.

Common formalisms If we look at the formalisms used (table 5), it appears that some are in the majority.

We can see that PVS and SAL are the most widely used specification languages. Over the 14 articles that use specification languages, we find that SAL is the most used with 5 articles using it. PVS is also widely used with 4 articles using it. Those two cover more than the half of the articles using specification language.

B and event-B models are still the most used for refinement processes. 6 articles present refinement processes and half of those use B and event-B models. We find 2 articles using Z and 1 article using μCharts.

New formalisms During this analysis, we have seen some formalisms close to the nomenclature we have set (see section 2.2). But other formalisms could not be easyly classified in one of the proposed families. We identified 8 papers that use ad-hoc formalisms or formalisms out of the nomenclature.

In those we find, for example, the formal definition of task models and use cases by using an ad-hoc formalism based on sets of partially ordered sets. We also find the modelling of several physical devices with a new and ad-hoc formalism. Another paper presents the formal definition of different emotions by using an ad-hoc formalism. An article presents security properties and the different means (access control language RW, ProVerif's query language, applied π-calculus) of formalising those.

Maturity of case studies A main case study is frequently presented: the "infusion pump" system. Other systems are presented and considered as "textbook" cases, representing more than half of the papers.

The infusion pump is a safety critical medical device and is used by 7 out of 43 articles. 3 of those study the data part of the whole system by modelling it and validate some properties on a sub-system only. 3 other articles study the full system. They model the final device or its specification in order to check whether the device or its specification validate the global requirements of infusion pump. The last article studies the possible interactions between a user and the system. They model those in the form of interaction sequences corresponding to the human user tasks.

This approach demonstrated the feasibility of the proposed methods but remains limited. We note that even if an infusion pump is a safety critical system, the studies made for this system do not necessarily address safety critical aspects. Indeed, only 3 articles focus on the full system and its certification oriented requirements. Only those demonstrate the scalability of the formalisms used.

16 out of 43 articles focus on "textbook" cases and address the user interface part (web application, smartphone application, e-voting system, etc.). Those allow authors to easily illustrate the use of several formal methods and the properties inherent to those. The systems are modelled, several properties, inherent to the formalisms or to the systems, are studied. However, these articles only illustrate the formal methods and do not allow authors to demonstrate the potential scalability of these formal methods.

5 Conclusion

Aim and contribution of this article The aim of this article is to review different research work on formal methods applied to interactive systems. The overall contribution is to provide a review of the literature, 43 articles, from the International Workshop on Formal Methods for Interactive Systems. This workshop took place seven times from 2006 to 2018. First we propose an analytical framework based on a few questions. Then we present several properties of interactive systems and classify them. We set a classic nomenclature of formalisms. This analytical framework is provided with an analysis grid of our own. Those highlight the following points: formalisms used, properties studied, case study used to illustrate the analysis. Finally, we highlight the findings and the outgoing issues.

Discussion Interactive systems are increasingly used in several sectors and propose several kinds of interactions with human users. The interactions can be from the system to the user by using sound notifications or display notifications in order to provide information to the user about the actual internal state of the system. They can also be from the user to the system with many interaction solutions such as mouse clicks, data entries with keyboards or buttons on the system or soft keyboards and buttons on the display of the system interface. All these interactions are source of new challenges when when the objective is to perform the formal verification and validation of their related properties.

During the last decade, a substantial work has been done in order to study how formalisms and methods can be applied to interactive system. A lot of them have demonstrated that it is possible to take into account a lot of classes of properties. High level properties such as those related to the tasks the user may accomplish or those related to the abstract interface have been studied. The classical formalisms relying on state and transition paradigm can be easily used to model these elements. However, properties related to the concrete part of the interface (involving characteristics of the graphical scene) remain largely uncovered by studies. As we highlighted in the section 4, we note that the properties related to the perceptibility have not yet been studied. This is not a real surprise: these properties require to model characteristics of the system which are not traditionaly handled by formal models: color of graphical objects, forms, dimension, visibility, collision etc. Modeling them remains a big challenge.

Perspectives Our research team works in the aeronautics sector. Thus, we focus on interactive and critical systems related to this sector. Interfaces with a very rich graphical scene are becoming increasingly important in aircraft cockpits. In this context, we develop a reactive language, Smala[1], allowing us to develop interfaces and interactions at the same level.

The issue related to visual perceptibility properties is then important in our opinion. In Béger et al. [24] we propose elements for formalising graphical properties. We set three basic properties that compose the node of our formalism: the display order depending on the display layer of graphical elements, the intersection depending on the domain of graphical elements and the colour equality. We also present a scene graph we extract from the Smala source code. It models interactive systems and their graphical interface in a new way. It also gives information about graphical elements and their variables (position, colour, opacity, etc.).

From those, we can formally define graphical requirements for an aeronautic system specified in a standard (ED-143 [1]). The formalism defines requirements such as the colour equality/inequality, the authorized/unauthorized positions and the display order. The scene graph defines requirements we can not write with our formalism such as the shape of graphical elements.

We aim at defining new graphical properties in order to express with our formalism requirements related to the shape and the relative positions of graphical elements. In order to automatically validate the requirements, we want to link our formalism to the Smala source code by using code annotations.

Acknowledgments

This work is partly funded by the ANR project FORMEDICIS, ANR-16-CE25-0007.

References

1. Ed 143 - minimum operational performance standards for traffic alert and collision avoidance system ii (tcas ii) (April 2013)
2. A Bargh, J.: The Four Horsemen of Automaticity: Awareness, Efficiency, Intention, and Control in Social Cognition., vol. 2 (01 1994)
3. Abrial, J.R.: The B-book: Assigning Programs to Meanings. Cambridge University Press, New York, NY, USA (1996)
4. Ament, M., Cox, A., Blandford, A., Brumby, D.: Working memory load affects device-specific but not task-specific error rate. CogSci 2010: Proceedings of the Annual Conference of the Cognitive Science Society pp. 91 – 96 (2010)
5. Anderson, H., Ciobanu, G.: Markov abstractions for probabilistic pi-calculus. Electronic Communications of the EASST **22** (01 2009)
6. Arapinis, M., Calder, M., Denis, L., Fisher, M., Gray, P., Konur, S., Miller, A., Ritter, E., Ryan, M., Schewe, S., Unsworth, C., Yasmin, R.: Towards the verification of pervasive systems. Electronic Communications of the EASST **22** (01 2009)
7. Baeten, J.: A brief history of process algebra. Theoretical Computer Science **335**(2), 131 – 146 (2005), process Algebra
8. Baier, C., Katoen, J.P.: Principles of Model Checking (Representation and Mind Series). The MIT Press (2008)
9. Banach, R., Razavi, J., Debicki, O., Mareau, N., Lesecq, S., Foucault, J.: Application of formal methods in the inspex smart systems integration project. In: FMIS 2018 (5 2018)

[1] http://smala.io/

10. Barbosa, M.A., Barbosa, L.S., Campos, J.C.: Towards a coordination model for interactive systems. Electronic Notes in Theoretical Computer Science **183**, 89 – 103 (2007), proceedings of the First International Workshop on Formal Methods for Interactive Systems
11. Barnett, M., and: The spec# programming system: An overview. In: CASSIS 2004, Construction and Analysis of Safe, Secure and Interoperable Smart devices. Lecture Notes in Computer Science, vol. 3362, pp. 49–69. Springer (January 2005)
12. Bass, E.J., Feigh, K.M., Gunter, E., Rushby, J.: Formal modeling and analysis for interactive hybrid systems **45** (01 2011)
13. Beaudouin-Lafon, M.: Designing interaction, not interfaces. In: Proceedings of the Working Conference on Advanced Visual Interfaces. pp. 15–22. AVI '04, ACM, New York, NY, USA (2004)
14. Beckert, B., Beuster, G.: Guaranteeing consistency in text-based human-computer-interaction (2007), proceedings of the First International Workshop on Formal Methods for Interactive Systems
15. Behrmann, G., David, A., Larsen, K.G.: A Tutorial on Uppaal, pp. 200–236. Springer Berlin Heidelberg, Berlin, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30080-9_7
16. Bhandal, C., Bouroche, M., Hughes, A.: A process algebraic description of a temporal wireless network protocol **45** (01 2011)
17. Bhattacharya, S., Basu, A., Samanta, D., Bhattacherjee, S., Srivatava, A.: Some issues in modeling the performance of soft keyboards with scanning (2007), proceedings of the First International Workshop on Formal Methods for Interactive Systems
18. Boldo, S., Lelay, C., Melquiond, G.: Formalization of Real Analysis: A Survey of Proof Assistants and Libraries. Mathematical Structures in Computer Science **26**(7), 1196–1233 (Oct 2016)
19. Bonnefon, J.F., Longin, D., Nguyen, M.H.: A logical framework for trust-related emotions. Electronic Communications of the EASST **22** (01 2009)
20. Bowen, J., Hinze, A.: Supporting mobile application development with model-driven emulation **45** (01 2011)
21. Bowen, J., Reeves, S.: Formal models for informal gui designs. Electronic Notes in Theoretical Computer Science **183**, 57 – 72 (2007), proceedings of the First International Workshop on Formal Methods for Interactive Systems
22. Bowen, J., Reeves, S.: Refinement for user interface designs. Electronic Notes in Theoretical Computer Science **208**, 5 – 22 (2008), proceedings of the 2nd International Workshop on Formal Methods for Interactive Systems
23. Bowen, J., Reeves, S.: Ui-design driven model-based testing. Electronic Communications of the EASST **22** (01 2009)
24. Béger, P., Becquet, V., Leriche, S., Prun, D.: Contribution à la formalisation des propriétés graphiques des systèmes interactifs pour la validation automatique. In: Afadl 2019, 18èmes journées Approches Formelles dans l'Assistance au Developpement de Logiciels . Toulouse, France (Jun 2019)
25. Bérard, B., et al.: Systems and Software Verification: Model-Checking Techniques and Tools. Springer Publishing Company, Incorporated, 1st edn. (2010)
26. Calder, M., Gray, P., Unsworth, C.: Tightly coupled verification of pervasive systems. Electronic Communications of the EASST **22** (01 2009)
27. Campos, J., Harrison, M.: Modelling and analysing the interactive behaviour of an infusion pump **45** (01 2011)
28. Cansell, D., Gibson, J.P., Méry, D.: Refinement: A constructive approach to formal software design for a secure e-voting interface. Electronic Notes in Theoretical Computer Science **183**, 39 – 55 (2007), proceedings of the First International Workshop on Formal Methods for Interactive Systems
29. Cartwright-Finch, U., Lavie, N.: The role of perceptual load in inattentional blindness. Cognition **102**(3), 321 – 340 (2007)

30. Cerone, A.: Closure and attention activation in human automatic behaviour: A framework for the formal analysis of interactive systems **45** (01 2011)
31. Cerone, A.: Towards a cognitive architecture for the formal analysis of human behaviour and learning. In: Mazzara, M., Ober, I., Salaün, G. (eds.) Software Technologies: Applications and Foundations. pp. 216–232. Springer International Publishing, Cham (2018)
32. Cerone, A., Elbegbayan, N.: Model-checking driven design of interactive systems. Electronic Notes in Theoretical Computer Science **183**, 3 – 20 (2007), proceedings of the First International Workshop on Formal Methods for Interactive Systems
33. Cerone, A., Zhao, Y.: Stochastic modelling and analysis of driver behaviour. ECEASST **69** (2013)
34. Cleaveland, R., Li, T., Sims, S.: The concurrency workbench of the new century. User's manual, SUNY at Stony Brook, Stony Brooke, NY, USA (2000)
35. David, R., Alla, H.: Discrete, Continuous, and Hybrid Petri Nets. Springer Publishing Company, Incorporated, 2nd edn. (2010)
36. Dittmar, A., Hübner, T., Forbrig, P.: Hops: A prototypical specification tool for interactive systems. In: Graham, T.C.N., Palanque, P. (eds.) Interactive Systems. Design, Specification, and Verification. pp. 58–71. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
37. Dittmar, A., Schachtschneider, R.: Lightweight interaction modeling in evolutionary prototyping. ECEASST **69** (2013)
38. Dix, A., Ghazali, M., Ramduny-Ellis, D.: Modelling devices for natural interaction. Electronic Notes in Theoretical Computer Science **208**, 23 – 40 (2008), proceedings of the 2nd International Workshop on Formal Methods for Interactive Systems
39. E. Raymond, J., Shapiro, K., Arnell, K.: Temporary suppression of visual processing in an rsvp task: An attentional blink? Journal of experimental psychology. Human perception and performance **18**, 849–60 (09 1992)
40. Geniet, R., Singh, N.K.: Refinement based formal development of human-machine interface. In: Mazzara, M., Ober, I., Salaün, G. (eds.) Software Technologies: Applications and Foundations. pp. 240–256. Springer International Publishing, Cham (2018)
41. Goldson, D., Reeve, G., Reeves, S.: μ-chart-based specification and refinement. In: Proceedings of the 4th International Conference on Formal Engineering Methods: Formal Methods and Software Engineering. pp. 323–334. ICFEM '02, Springer-Verlag, Berlin, Heidelberg (2002)
42. Goranko, V., Galton, A.: Temporal logic. *The Stanford Encyclopedia of Philosophy* (Winter 2015 Edition), Edward N. Zalta (ed.) (2015)
43. Gosain, A., Sharma, G.: Static analysis: A survey of techniques and tools. In: Intelligent Computing and Applications. pp. 581–591. Springer India, New Delhi (2015)
44. Harrison, M.D., Kray, C., Campos, J.C.: Exploring an option space to engineer a ubiquitous computing system. Electronic Notes in Theoretical Computer Science **208**, 41 – 55 (2008), proceedings of the 2nd International Workshop on Formal Methods for Interactive Systems
45. Harrison, M.D., Masci, P., Campos, J.C., Curzon, P.: Automated theorem proving for the systematic analysis of an infusion pump. ECEASST **69** (2013)
46. Harrison, M.D., Masci, P., Campos, J.C.: Formal modelling as a component of user centred design. In: Mazzara, M., Ober, I., Salaün, G. (eds.) Software Technologies: Applications and Foundations. pp. 274–289. Springer International Publishing, Cham (2018)
47. Hillston, J.: A Compositional Approach to Performance Modelling. Cambridge University Press, New York, NY, USA (1996)
48. Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1985)

49. Holzmann, G.: The SPIN Model Checker: Primer and Reference Manual. Addison-Wesley Professional, 1st edn. (2011)
50. Huang, H., Rukšėnas, R., Ament, M., Curzon, P., Cox, A., Blandford, A., Brumby, D.: Capturing the distinction between task and device errors in a formal model of user behaviour **45** (01 2011)
51. ISO-8807:1989: Information processing systems - open systems interconnection - lotos - a formal description technique based on the temporal ordering of observational behaviour (1989)
52. Johnson, C.W.: Using assurance cases and boolean logic driven markov processes to formalise cyber security concerns for safety-critical interaction with global navigation satellite systems **45** (01 2011)
53. Kray, C., Kortuem, G., Krüger, A.: Adaptive navigation support with public displays. In: Proceedings of the 10th International Conference on Intelligent User Interfaces. pp. 326–328. IUI '05, ACM, New York, NY, USA (2005)
54. Leriche, S., Conversy, S., Picard, C., Prun, D., Magnaudet, M.: Towards handling latency in interactive software. In: Mazzara, M., Ober, I., Salaün, G. (eds.) Software Technologies: Applications and Foundations. pp. 233–239. Springer International Publishing, Cham (2018)
55. Masci, P., Curzon, P., Blandford, A., Furniss, D.: Modelling distributed cognition systems in pvs **45** (01 2011)
56. Masci, P., Rukšėnas, R., Oladimeji, P., Cauchi, A., Gimblett, A., Li, Y., Curzon, P., Thimbleby, H.: On formalising interactive number entry on infusion pumps **45** (01 2011)
57. Mori, G., Paterno, F., Santoro, C.: Design and development of multidevice user interfaces through multiple logical descriptions. IEEE Transactions on Software Engineering **30**(8), 507–520 (Aug 2004). https://doi.org/10.1109/TSE.2004.40
58. de Moura, L., Owre, S., Rueß, H., Rushby, J., Shankar, N., Sorea, M., Tiwari, A.: Sal 2. In: Alur, R., Peled, D.A. (eds.) Computer Aided Verification. pp. 496–500. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
59. Myers, B.A., Rosson, M.B.: Survey on user interface programming. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. pp. 195–202. CHI '92, ACM, New York, NY, USA (1992)
60. Navarre, D., Palanque, P., Ladry, J.F., Barboni, E.: Icos: A model-based user interface description technique dedicated to interactive systems addressing usability, reliability and scalability. ACM Trans. Comput.-Hum. Interact. **16**(4), 18:1–18:56 (Nov 2009)
61. Norman, G., Palamidessi, C., Parker, D., Wu, P.: Model checking the probabilistic π-calculus. In: Proc. 4th International Conference on Quantitative Evaluation of Systems (QEST'07). pp. 169–178. IEEE Computer Society (2007)
62. Oladimeji, P., Masci, P., Curzon, P., Thimbleby, H.: Pvsio-web: a tool for rapid prototyping device user interfaces in pvs. ECEASST **69** (2013)
63. Owicki, S., Lamport, L.: Proving liveness properties of concurrent programs. ACM Trans. Program. Lang. Syst. **4**(3), 455–495 (Jul 1982)
64. Puschner, P., Burns, A.: A review of worst-case execution-time analyses. Real-time Systems - RTS (01 1999)
65. Rukšėnas, R., Back, J., Curzon, P., Blandford, A.: Formal modelling of salience and cognitive load. Electronic Notes in Theoretical Computer Science **208**, 57 – 75 (2008), proceedings of the 2nd International Workshop on Formal Methods for Interactive Systems
66. Rukšėnas, R., Curzon, P., Blandford, A.: Detecting cognitive causes of confidentiality leaks. Electronic Notes in Theoretical Computer Science **183**, 21 – 38 (2007), proceedings of the First International Workshop on Formal Methods for Interactive Systems
67. Rukšėnas, R., Curzon, P.: Abstract models and cognitive mismatch in formal verification **45** (01 2011)

68. Rukšėnas, R., Masci, P., Harrison, M.D., Curzon, P.: Developing and verifying user interface requirements for infusion pumps: A refinement approach. ECEASST **69** (2013)
69. Ryan, M.D., Smyth, B.: Applied pi calculus. In: Cortier, V., Kremer, S. (eds.) Formal Models and Techniques for Analyzing Security Protocols, chap. 6. IOS Press (2011)
70. Sabelfeld, A., Myers, A.C.: Language-based information-flow security. IEEE J.Sel. A. Commun. **21**(1), 5–19 (Sep 2006)
71. SC-205, R.F., 71, E.A.W.G.: Rtca/do-178c software considerations in airborne systems and equipment certification (December 2011)
72. SC-205, R.F., 71, E.A.W.G.: Rtca/do-333 formal methods supplement to do-178c and do-278a (December 2011)
73. Shankar, N.: Pvs: Combining specification, proof checking, and model checking. In: Srivas, M., Camilleri, A. (eds.) Formal Methods in Computer-Aided Design. pp. 257–264. Springer Berlin Heidelberg, Berlin, Heidelberg (1996)
74. Silva, J.L., Campos, J.C., Paiva, A.C.: Model-based user interface testing with spec explorer and concurtasktrees. Electronic Notes in Theoretical Computer Science **208**, 77 – 93 (2008), proceedings of the 2nd International Workshop on Formal Methods for Interactive Systems
75. Silva, J.L., Fayollas, C., Hamon, A., Palanque, P., Martiinie, C., Barboni, E.: Analysis of wimp and post wimp interactive systems based on formal specification. ECEASST **69** (2013)
76. Sinnig, D., Chalin, P., Khendek, F.: Towards a common semantic foundation for use cases and task models. Electronic Notes in Theoretical Computer Science **183**, 73 – 88 (2007), proceedings of the First International Workshop on Formal Methods for Interactive Systems
77. Soukoreff, R.W., Mackenzie, I.S.: Theoretical upper and lower bounds on typing speed using a stylus and a soft keyboard. Behaviour & Information Technology **14**(6), 370–379 (1995)
78. Spivey, J.M.: The Z Notation: A Reference Manual. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1989)
79. Standardization, I.: ISO 9241-11: Ergonomic Requirements for Office Work with Visual Display Terminals (VDTs): Part 11: Guidance on Usability (1998)
80. Su, L., Bowman, H., Barnard, P.: Performance of reactive interfaces in stimulus rich environments, applying formal methods and cognitive frameworks. Electronic Notes in Theoretical Computer Science **208**, 95 – 111 (2008), proceedings of the 2nd International Workshop on Formal Methods for Interactive Systems
81. Thimbleby, H., Gimblett, A.: Dependable keyed data entry for interactive systems **45** (01 2011)
82. Turner, J., Bowen, J., Reeves, S.: Using Abstraction with Interaction Sequences for Interactive System Modelling: STAF 2018 Collocated Workshops, Toulouse, France, June 25-29, 2018, Revised Selected Papers, pp. 257–273 (06 2018)
83. Westergaard, M.: A game-theoretic approach to behavioural visualisation. Electronic Notes in Theoretical Computer Science **208**, 113 – 129 (2008), proceedings of the 2nd International Workshop on Formal Methods for Interactive Systems

Preliminary Thoughts on User Interfaces for Logic-based Medical Image Analysis

Vincenzo Ciancia$^{1[0000-0003-1314-0574]}$ and Mieke Massink$^{1[0000-0001-5089-002X]}$

Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo"
Consiglio Nazionale delle Ricerche
{vincenzo.ciancia,mieke.massink}@isti.cnr.it

Abstract. Spatial logic and spatial model checking may provide domain experts with a convenient and very concise way to specify contouring and segmentation operations, grounded on the solid mathematical foundations of Topological Spatial Logics. Recent results in the domain of brain tumour segmentation have shown that this approach has the potential to reach comparable accuracy and computational efficiency as other state-of-the-art techniques in this domain. One of the open challenges, specific for this approach, is how such a technique could be best embedded into the common workflow of clinicians and radiologists. In particular, domain experts involved in research would profit from a set-up and user interface that would also support collaboration and exchange of expertise among medical physicists, engineers and technicians. We briefly describe some related work, setting, aims, and challenges for such an integrated interface.

Keywords: Spatial Logics · Model Checking · Human-Computer Interaction · Medical Imaging

1 Introduction

Computational methods play a prominent role in many applications for medical image analysis. An important example in the domain of neuro-imaging is segmentation of the human brain and of brain tumours. This constitutes a very active research area (see for example [18,10,12,22,25] and references therein) in which many automatic and semi-automatic methods have been proposed to overcome the current time consuming practise of manual delineation of brain tumours in magnetic resonance images (MRI) and at providing an accurate, reliable and reproducible method of segmentation of the tumour area and related tissues [12].

In recent work [5,4,2,6,7] by a research group (including the authors) of experts in model checking and in medical physics, a novel approach to image segmentation has been developed, aimed at merging the capabilities of state-of-the-art libraries of computational imaging algorithms with the unique combination of *declarative specification* and optimised execution provided by *spatial logic model* *checking*. A declarative approach makes analysis transparent, reproducible and human readable. Moreover, the specifications are concise and can easily be exchanged by domain experts. This novel methodology exploits the *relative* spatial relations between tissues of interest, mentioned earlier, and encompasses different segmentation methods, such as texture features, local histogram processing, and prior knowledge, that can be freely combined and nested, since they are mapped to operators of a domain-specific language for image analysis [3].

The approach, and its related (command line) tool `VoxLogicA`[1], has been applied to an extensive existing benchmark of medical images for the segmentation of brain tumours [20] and preliminary results show that it can reach state-of-the-art accuracy, compared to many best performing algorithms in the field [6] which, nowadays, are mostly dominated by deep learning based approaches [20,19]. The tools and methods can be used both for two-dimensional (2D) and three-dimensional (3D) medical images such as those produced by MRI machines. The use of 3D information may lead to improved accuracy and is of high interest in the field.

The logical-declarative approach of `VoxLogicA` permits users to write very short "queries" that exploit familiar, basic spatial concepts such as region proximity, reachability, separation, distance and similarity, arbitrarily nested and combined with traditional Boolean operators (union, intersection, complementation) and derived and domain specific operators for medical imaging (e.g. contact, region growing, filter, percentiles). Such queries are then turned into optimised parallel execution pipelines using the internal execution engine of VoxLogicA (in essence, a memoizing global spatial model checker), and can be run over arbitrarily large datasets. In Figure 1 we show an example (the specification in [6]) of the current (textual) type of logical specifications and the input, namely a 3D image of a human brain affected by *glioblastoma*, and output of the analysis (note that we are only showing a slice of the input and output, but the analysis is carried out in 3D). We refer the reader to [6] for further details.

Among the planned future developments is the inclusion of means for interactive refinement of analysis such as the fine-tuning of specific values (e.g. thresholds or distances) and the embedding of the analysis tools in the common workflow and related digital space of domain experts.

Even if current results obtained using `VoxLogicA` appear promising, the tool lacks a user-oriented graphical interface that can leverage adoption in the medical imaging community. This brings us to the main question that we investigate in this paper, namely what could a suitable graphical user interface look like, that smoothly integrates the proposed analysis method with other functionalities that are essential for the professional needs of domain experts, such as the display of high-resolution medical images, reporting facilities, version control, and visual design of analysis. What characteristics should such an interface have so that domain experts can perform their inherently multi-tasking work smoothly, minimising the cognitive costs of reduced comprehension and recall and related diagnostic error rates?

[1] `VoxLogicA: https://github.com/vincenzoml/VoxLogicA.`

```
let background = touch(intensity <.  0.1,border)
let brain = !background

let pflair = percentiles(intensity,brain)
let hI = pflair >.  0.95
let vI = pflair >.  0.86
let hyperIntense = flt(5.0,hI)
let veryIntense = flt(2.0,vI)

let growTum = grow(hyperIntense,veryIntense)
let tumSim = similarTo(growTum)
let tumStatCC = flt(2.0,tumSim >.  0.6)
let gtv = grow(growTum,tumStatCC)
```

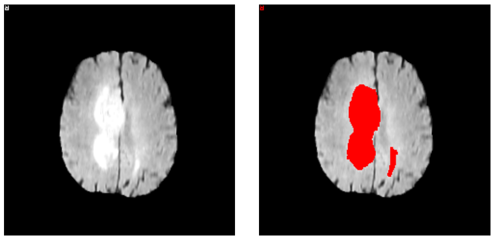


Fig. 1. The spatial logic specification identifying glioblastoma for radiotherapy purposes, and one example of input (left image) and output (right image), where all points in the image that satisfy the spatial logic formula `gtv` are shown in red (`gtv` stands for gross tumour volume, i.e. the part of the tumour that can actually be seen in an MRI image). Slice from 3D image Brats17_2013_2_1, FLAIR, from the Brain Tumour Image Segmentation Benchmark BraTS 2017 [23,1]

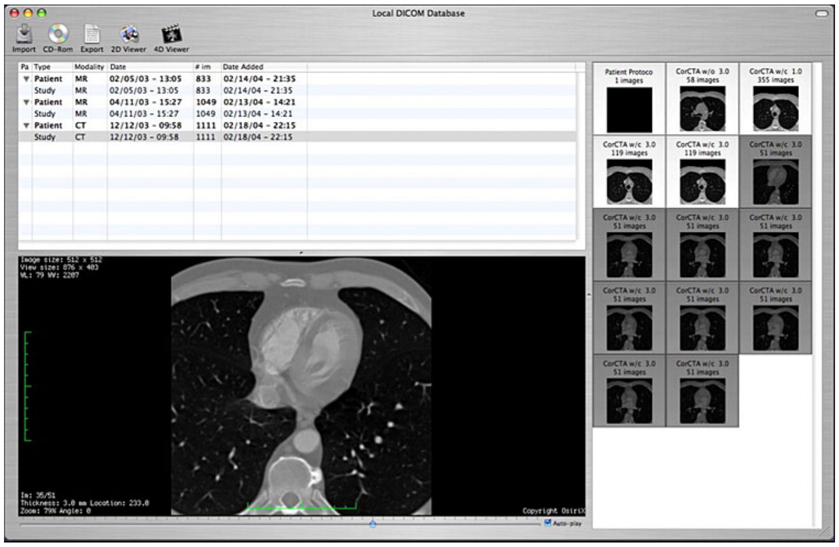
2 Related Work and Literature Review

Although systems for the automatic analysis of medical images have been a topic of study from the early 70ties and have since then found increasingly wide application in the clinical setting, there are surprisingly few studies on the usability aspects of such systems. Some studies have addressed the ergonomic and physical aspects of the radiologist's workspace (see some references in [21]).

Also some information on workspace use and preference of radiologists is reported in [21] and is based on a brief survey involving 336 respondents among radiology professionals. The report provides some insight in the preferred workstation setups (such as a strong preference for having two diagnostic monitors and a divergent opinion on the preferred number of additional non-diagnostic monitors) and in the number and kind of applications that are kept open simultaneously during diagnosis activity.

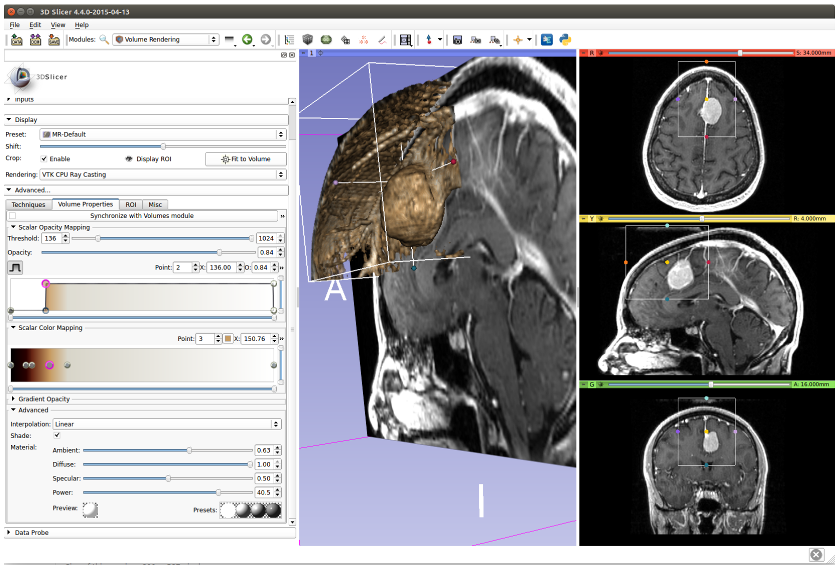
A recent review of existing and potential computer user interfaces for radiology can be found in [16]. This article underlines the importance of good usability and suggests that diagnostic imaging needs user interfaces to deal both with image manipulation and workflow management. Various interfaces for working with 2D and 3D medical images are discussed. The latter are deemed preferable for the added value these may provide. In [8] a comparative usability study is performed for a series of tools for drawing Regions of Interest (ROI) on medical images. The comparison is based on a use case analysis.

Graphical user interfaces for medical imaging typically focus on navigation of a database of images (see e.g. Figure 2), with some notable exceptions. 3D-Slicer (Figure 3) is aimed at interactive execution of analysis programs written in *python* or *C++*, letting the user calibrate the parameters of the analysis.



Pa	Type	Modality	Date	# im	Date Added
▼	Patient	MR	02/05/03 - 13:05	833	02/14/04 - 21:35
	Study	MR	02/05/03 - 13:05	833	02/14/04 - 21:35
▼	Patient	MR	04/11/03 - 15:27	1049	02/13/04 - 14:21
	Study	MR	04/11/03 - 15:27	1049	02/13/04 - 14:21
▼	Patient	CT	12/12/03 - 09:58	1111	02/18/04 - 22:15
	Study	CT	12/12/03 - 09:58	1111	02/18/04 - 22:15

Fig. 2. The OsiriX medical image viewer, with database navigation capabilities (see https://www.osirix-viewer.com/).



Modules: Volume Rendering
Preset: MR-Default
Crop: ✓ Enable Display ROI Fit to Volume
Rendering: VTK CPU Ray Casting
Threshold: 136 1024
Opacity: 0.84
Point: 2 X: 136.00 O: 0.84
Point: 3 X: 150.76
Interpolation: Linear
Shade: ✓
Material: Ambient: 0.63
Diffuse: 1.00
Specular: 0.50
Power: 40.5
Preview: Presets:

Fig. 3. The 3d Slicer tool, integrating a medical image viewer with execution of user programs and interactive calibration of parameters (see https://www.slicer.org/).

In [11] a proof-of-concept visual programming interface to the imaging library ITK[2] has been designed using Simulink[3]. Along the same lines are the efforts in [17]. MITK [24], MeVisLab [15] and SCIRun are software tools that encompass visual dataflow programming for scientific purposes and image analysis, most of them also based on ITK. Although the tool VoxLogicA is also based on ITK, the level of abstraction is quite different, as VoxLogicA does not directly expose the methods of the underlying imaging library (whose programming interface is aimed at specialists of the field), but rather hides them under the *lingua franca* of spatial logics, which is aimed at usability by a wider, less-technical audience (the difference, in spirit, is the same that exist between analysing data using imperative programming languages such as C++ or Python, versus using declarative approaches like the ubiquitous *structured query language* SQL).

Finally, very close to our current interest is the work by Gambino et al. [13] that proposes a framework for data-driven adaptive graphical user interface generation. In this framework the user can insert or remove tools directly at runtime in a simple and immediate way. The software architecture is based on the well-known Model-View-Controller (MVC) paradigm to guarantee a modular and scalable set-up that can be extended when needed. The work provides detailed interaction diagrams of the proposed GUI and it has been implemented as a plug-in of the DICOM[4] viewer OsiriX.

3 Computer-assisted Design of Spatial Logic Specifications for Medical Imaging

Our effort is aimed at incorporating spatial logic model checking in the classical interfaces for medical images viewers, such as OsiriX, and amalgamating these with a visual programming interface for logical specifications, with interactive calibration of parameters and navigational visualisation of the results on possibly large datasets of medical imaging cases.

Although a typical spatial logic specification for image analysis is rather concise (in general it consists of a few dozens of lines of text), it may be very helpful for a domain expert to have quick, detailed and visual feedback on all intermediate formulas. This could help considerably in letting such experts quickly develop a feeling for the effect of various basic and derived operators and more abstract building blocks of the spatial logic and also in an easier exploration of the effect of numeric parameters of operators such as distance, correlation coefficients and thresholds. Also the stability of more abstract building blocks is an issue: Does the specification give satisfactory results for all images of a given data (image) set? Furthermore, providing such kind of visual feedback could considerably reduce the learning curve for the uptake of the approach also by domain experts with a more clinical background and less skilled in programming.

[2] The Insight Segmentation and Registration Toolkit, see `https://www.itk.org/`.
[3] See https://www.mathworks.com/products/simulink.html
[4] DICOM stands for "Digital Imaging and COmmunications in Medicine" and is a widely used standard for digital medical images and related meta data.

We envisage *computer-assisted design* of logical specification as an instrument to address such issues, and leverage adoption of formal methods in novel application fields. Computer-assisted logical specification crucially depends on the design of appropriate *user interfaces*, aimed at *discoverability* of useful design patterns, and of potential mistakes, in order to make logical specification an engaging task for domain experts in specific application fields. We shall now discuss some of the major challenges we identified in this scenario.

Let us first establish some key concepts. Making a step towards abstraction, the syntax of a logic formula is a tree in which each node is a logical operator, and each child corresponds to an argument, which, inductively, can be a logic formula. Leaf nodes obviously correspond to constants or atomic propositions. A logical specification is therefore a *forest* of syntax trees, one for each formula to be checked. As each analysis is typically meant to be applied to several models (in this case medical images), one may also consider the set of all models to be analysed as a *data stream*, and consider each node of the syntax tree as a processing node in a processing pipeline.

The visual design of such kinds of processing pipelines is part of the research line on *dataflow programming* [9]. Basically, each processing unit is represented as a node in a diagram/graph, and arcs represent the flow of data. Such type of design is common, for instance, in programming environments for sound design (for instance, *Pure Data*[5]). In the case of logic formulas, each processing node is a logical operator, and arcs go from the bottom of the syntax tree to the top, linking each argument to its parent operator.

3.1 Challenges

In the particular case of dataflow programs represented by spatial logic specifications, and especially when such logical specification is aimed at *model checking*, some peculiar issues related to Human-Computer Interaction become apparent. In the following we attempt to discuss some of the most relevant ones we identified. The design issues we discuss are common to many logical formalisms and case studies, modulo the fact that the visualisation of intermediate and final results may not be as immediate as in the case of images. For instance, in a traditional temporal model checker, visualisation can consist of the Boolean satisfaction value of a formula on the given system model, but also, for debugging purposes, the value of sub-formulas on relevant states, or a counterexample when a given formula does not hold. Nevertheless, from now on, we focus on the main case study of the paper, namely spatial logics for medical imaging, drawing expertise from our recent research [6,7].

Several dimensions of the design space. The design space of the analysis is shaped by several orthogonal dimensions. First and foremost, one may think of the numerical parameters of formulas (e.g., thresholds, or distances). But when it comes to interactive development of a set of logic formulas, there is much more,

[5] See `https://puredata.info/`.

laying "outside" of the specification itself. An important dimension to consider is that of the *dataset* (i.e. the set of medical images and related meta-data). Each analysis is meant to be tested against some specific datasets, and then to be reused on possibly larger ones. For each dataset, there is some information which is gathered "globally", that is, extracted from the analysis after it has been run on the whole dataset, such as accuracy scores, and other information coming from the local results of the analysis (both the final result, and that of relevant intermediate steps) for each specific element of the dataset. Specification designers need to visualise such results on more than one dataset at the same time, in order to receive immediate feedback on possible improvements or design mistakes. Another dimension is that of multiple *versions* of the same analysis, either due to incremental design by the same designer, or by the presence of multiple collaborators. In this case, not only it is very important to be able to see the result of multiple versions at the same time, but also to be able to *overlay* the local results of several versions and to clearly visualise the differences directly from within the user interface.

Textual representation vs. visualisation. Logic formulas are traditionally written in textual form, and it is not the purpose of our current research to replace textual information entirely with visual design. Instead, we envisage a user interface where it is possible to seamlessly edit both the visual and the textual representation of the specification. However, the two views may not be easily matched. For instance, maintaining coherency is difficult when the user cuts and pastes textual formulas from another source; visual information (especially layout) may be difficult to transmit as it is heavily context-dependent. Also, changes to text (e.g. renaming of variables) may render the visual representation inconsistent.

Layout vs. structure. Another tension in the design of user interfaces for logical specifications is that between layout of elements, and syntactic structuring of sub-formulas. As we mentioned briefly, a formula is defined by its syntax tree, which is indeed a graph. However, the visual rendering of such a tree is ambiguous. For instance, should a block of formulas in a *let binding* such as `let x = formula1 AND formula2` be treated as a subgraph of the dataflow program, with `formula1` and `formula2` visible, or as a black-box entity named x that can eventually be explored to change the internals of the block? Indeed, the possibility of *refactoring* the textual presentation (e.g. substituting x with its definition) makes this aspect particularly complex.

Equivalent subgraphs. Strictly linked to the previous point is the fact that model checkers typically adopt methods, such as *memoization*, to identify equivalent sub-formulas and avoid to recompute them. Although this is a pure implementation aspect in principle, it may be important to make such techniques *observable* in the user interface, both as tools to simplify the design, and as a visual aid to help the user to manage the complexity of a specific design by identifying the components that perform the same tasks.

Non-linear effects. Classical examples of dataflow programming such as audio design typically enjoy a top-to-bottom dataflow and an incremental design process, in which the quality of the result is not generally affected by small mistakes in intermediate steps, as the processed data is constituted by continuous signals. On the other hand, logical specifications inherently deal with *discrete* data, and small mistakes may have *non-linear* effects, that propagate to the final result in unexpected, often subtle ways. Therefore, it is very important to design user interfaces that support easy, arbitrary visual rearranging of processing nodes in order to quickly visualise the effect of even minor changes, on "distant" intermediate steps. In current dataflow-based user interfaces, manual rearranging of nodes is tedious and not intended as a primary functionality.

User-centred Design. Various groups of users could be expected to use interfaces for the analysis of medical images. One group of users could be domain experts with a more technical background, such as radiologists and medical-physicists. They could be interested in the design of new specifications for particular kinds of analysis involving different parts of the body or with specific characteristics. Another class of users could be medical staff that need to review or make a diagnosis of a series of analyses and perhaps annotate these and produce reports in preparation of further patient treatment. These classes of users would use the interface in quite different ways, but providing a single integrated system could be helpful for a smooth collaboration and exchange of expertise and feedback between different kind of users. The needs of the various user groups should be carefully studied, in particular there may be concerns on cognitive aspects such as information overload, multi-tasking and focus in the critical task of medical image analysis for radiotherapy. It would be very interesting to investigate how user-centred design methods that combine contextual design and scenario based techniques with formal models, such as that proposed in [14], could be used in this setting.

4 Discussion

Systems for the automatic analysis of medical images have been an active topic since the early 70ties and have a wide uptake in the clinical setting. Nevertheless, there appears to be very little research on the design of suitable user interfaces for such system, despite the critical and cognitively demanding nature of the work. Professionals that analyse medical images on a regular basis need to manage much information of very different nature such as case information, medical records, dictation systems, reporting facilities, literature search, access to databases and version control, to mention a few [21]. This requires an adequate support for forms of multi-tasking with minimal impact on the focus and already demanding cognitive load of domain experts. The integration of novel and more interactive approaches to medical imaging, such as those based on spatial logic and model checking or the execution of user programs and interactive calibration of parameters, pose further challenges on the interface design.

We have discussed some selected literature, preliminary ideas and further challenges as a first step towards a more detailed interface design for this domain of application.

References

1. Bakas, S., Akbari, H., Sotiras, A., Bilello, M., Rozycki, M., Kirby, J.S., Freymann, J.B., Farahani, K., Davatzikos, C.: Advancing the cancer genome atlas glioma MRI collections with expert segmentation labels and radiomic features. Scientific Data **4** (2017). https://doi.org/10.1038/sdata.2017.117, `https://doi.org/10.1038/sdata.2017.117`, online publication date: 2017/09/05
2. Banci Buonamici, F., Belmonte, G., Ciancia, V., Latella, D., Massink, M.: Spatial Logics and Model Checking for Medical Imaging. International Journal on Software Tools for Technology Transfer (Feb 2019). https://doi.org/10.1007/s10009-019-00511-9, `https://doi.org/10.1007/s10009-019-00511-9`, online First
3. Banci Buonamici, F., Belmonte, G., Ciancia, V., Latella, D., Massink, M.: Spatial logics and model checking for medical imaging. International Journal on Software Tools for Technology Transfer **Online First** (2019). https://doi.org/https://doi.org/10.1007/s10009-019-00511-9
4. Belmonte, G., Ciancia, V., Latella, D., Massink, M., Biondi, M., De Otto, G., Nardone, V., Rubino, G., Vanzi, E., Banci Buonamici, F.: A topological method for automatic segmentation of glioblastoma in mr flair for radiotherapy - ESMRMB 2017, 34th annual scientific meeting. Magnetic Resonance Materials in Physics, Biology and Medicine **30**(S1), 437 (oct 2017). https://doi.org/10.1007/s10334-017-0634-z, `https://doi.org/10.1007/s10334-017-0634-z`
5. Belmonte, G., Ciancia, V., Latella, D., Massink, M.: From collective adaptive systems to human centric computation and back: Spatial model checking for medical imaging. In: ter Beek, M.H., Loreti, M. (eds.) Proceedings of the Workshop on FORmal methods for the quantitative Evaluation of Collective Adaptive SysTems, FORECAST@STAF 2016, Vienna, Austria, 8 July 2016. EPTCS, vol. 217, pp. 81–92 (2016). https://doi.org/10.4204/EPTCS.217.10, `https://doi.org/10.4204/EPTCS.217.10`
6. Belmonte, G., Ciancia, V., Latella, D., Massink, M.: Voxlogica: A spatial model checker for declarative image analysis. In: Vojnar, T., Zhang, L. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part I. Lecture Notes in Computer Science, vol. 11427, pp. 281–298. Springer (2019). https://doi.org/10.1007/978-3-030-17462-0_16, `https://doi.org/10.1007/978-3-030-17462-0_16`
7. Belmonte, G., Ciancia, V., Latella, D., Massink, M.: Innovating medical image analysis via spatial logics. In: ter Beek, M.H., Fantechi, A., Semini, L. (eds.) From Software Engineering to Formal Methods and Tools, and Back. Lecture Notes in Computer Science (Accepted for publication, 2019)
8. Chen, C., Abdelnour-Nocera, J.L., Wells, S., Pan, N.: Usability practice in medical imaging application development. In: Holzinger, A., Miesenberger, K. (eds.) HCI and Usability for e-Inclusion, 5th Symposium of the Workgroup Human-Computer Interaction and Usability Engineering of the Austrian Computer Society, USAB 2009, Linz, Austria, November 9-10, 2009 Proceedings. Lecture Notes in Computer

Science, vol. 5889, pp. 405–415. Springer (2009). https://doi.org/10.1007/978-3-642-10308-7_29, https://doi.org/10.1007/978-3-642-10308-7_29

9. Dennis, J.B.: First version of a data flow procedure language. In: Programming Symposium, Proceedings Colloque Sur La Programmation. pp. 362–376. Springer-Verlag, Berlin, Heidelberg (1974), http://dl.acm.org/citation.cfm?id=647323.721501
10. Despotović, I., Goossens, B., Philips, W.: MRI segmentation of the human brain: Challenges, methods, and applications. Computational and Mathematical Methods in Medicine **2015**, 1–23 (2015). https://doi.org/10.1155/2015/450341, http://dx.doi.org/10.1155/2015/450341
11. Dickinson, A.W.L., Abolmaesumi, P., Gobbi, D.G., Mousavi, P.: SimITK: Visual programming of the ITK image-processing library within simulink. Journal of Digital Imaging **27**(2), 220–230 (Jan 2014). https://doi.org/10.1007/s10278-013-9667-7, https://doi.org/10.1007/s10278-013-9667-7
12. Dupont, C., Betrouni, N., Reyns, Vermandel, M.: On image segmentation methods applied to glioblastoma: State of art and new trends. IRBM **37**(3), 131–143 (jun 2016). https://doi.org/10.1016/j.irbm.2015.12.004, https://doi.org/10.1016%2Fj.irbm.2015.12.004
13. Gambino, O., Rundo, L., Cannella, V., Vitabile, S., Pirrone, R.: A framework for data-driven adaptive GUI generation based on DICOM. Journal of Biomedical Informatics **88**, 37–52 (2018). https://doi.org/10.1016/j.jbi.2018.10.009, https://doi.org/10.1016/j.jbi.2018.10.009
14. Harrison, M.D., Masci, P., Campos, J.C.: Formal modelling as a component of user centred design. In: Mazzara, M., Ober, I., Salaün, G. (eds.) Software Technologies: Applications and Foundations - STAF 2018 Collocated Workshops, Toulouse, France, June 25-29, 2018, Revised Selected Papers. Lecture Notes in Computer Science, vol. 11176, pp. 274–289. Springer (2018). https://doi.org/10.1007/978-3-030-04771-9_21, https://doi.org/10.1007/978-3-030-04771-9_21
15. Heckel, F., Schwier, M., Peitgen, H.O.: Object-oriented application development with mevislab and python. In: Fischer, S., Maehle, E., Reischuk, R. (eds.) GI Jahrestagung. LNI, vol. 154, pp. 1338–1351. GI (2009), http://dblp.uni-trier.de/db/conf/gi/gi2009.html#HeckelSP09
16. Ianessi, A., Marcy, P.Y., Clatz, O., Bertrand, A.S., Sugimoto, M.: A review of existing and potential computer user interfaces for modern radiology. Insights into imaging **9**(4), 599–609 (2018). https://doi.org/10.1007/s13244-018-0620-7
17. Le, H.D.K., Li, R., Ourselin, S., Potter, J.: A visual dataflow language for image segmentation and registration. In: Filipe, J., Shishkov, B., Helfert, M., Maciaszek, L.A. (eds.) Software and Data Technologies. pp. 60–72. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
18. Lemieux, L., Hagemann, G., Krakow, K., Woermann, F.: Fast, accurate, and reproducible automatic segmentation of the brain in t1-weighted volume mri data. Magnetic Resonance in Medicine **42**(1), 127–135 (1999)
19. Litjens, G.J.S., Kooi, T., Bejnordi, B.E., Setio, A.A.A., Ciompi, F., Ghafoorian, M., van der Laak, J.A.W.M., van Ginneken, B., Sánchez, C.I.: A survey on deep learning in medical image analysis. Medical Image Analysis **42**, 60–88 (2017). https://doi.org/10.1016/j.media.2017.07.005, https://doi.org/10.1016/j.media.2017.07.005
20. Menze, B., et al.: The multimodal brain tumor image segmentation benchmark (brats). IEEE Transactions on Medical Imaging **34**(10), 1993–2024 (2015). https://doi.org/10.1109/TMI.2014.2377694

21. Sharma, A., Wang, K., Siegel, E.: Radiologist digital workspace use and preference: a survey-based study. J. Digit. Imaging **30**(6), 687–694 (2017). https://doi.org/10.1007/s10278-017-9971-8
22. Simi, V., Joseph, J.: Segmentation of glioblastoma multiforme from MR images – a comprehensive review. The Egyptian Journal of Radiology and Nuclear Medicine **46**(4), 1105–1110 (dec 2015). https://doi.org/10.1016/j.ejrnm.2015.08.001, `https://doi.org/10.1016%2Fj.ejrnm.2015.08.001`
23. Spyridon (Spyros) Bakas et al. (Ed.): 2017 international MICCAI BraTS Challenge: pre-conference proceedings (Sept 2017), `https://www.cbica.upenn.edu/sbia/Spyridon.Bakas/MICCAI_BraTS/MICCAI_BraTS_2017_proceedings_shortPapers.pdf`
24. Wolf, I., Vetter, M., Wegner, I., Nolden, M., Bottger, T., Hastenteufel, M., Schobinger, M., Kunert, T., Meinzer, H.P.: The medical imaging interaction toolkit (MITK): a toolkit facilitating the creation of interactive software by extending VTK and ITK. In: Robert L. Galloway, J. (ed.) Medical Imaging 2004: Visualization, Image-Guided Procedures, and Display. SPIE (May 2004). https://doi.org/10.1117/12.535112, `https://doi.org/10.1117/12.535112`
25. Zhu, Y., Young, G., Xue, Z., Huang, R., You, H., Setayesh, K., Hatabu, H., Cao, F., Wong, S.: Semi-automatic segmentation software for quantitative clinical brain glioblastoma evaluation. Academic Radiology **19**(8), 977–985 (aug 2012). https://doi.org/10.1016/j.acra.2012.03.026, `https://doi.org/10.1016%2Fj.acra.2012.03.026`

Synthesizing Glue Code for Graphical User Interfaces from Formal Specifications*

Contributed Talk based On Published Results

Keerthi Adabala and Rüdiger Ehlers

Clausthal University of Technology, Clausthal-Zellerfeld, Germany

Abstract. The glue code of a graphical user interface (GUI) is responsible for reacting to user input by changing the state of the user interface and triggering the computation to be performed. Here, we present an approach to synthesize such glue code. Our approach combines several ideas that work best in combination. We start from specifications in linear temporal logic (LTL) that express the desired interaction between the UI and the backend of the program. Our synthesis approach applies ideas from automata theory to the special case of user interfaces. We demonstrate that the approach is already scalable enough for first practical use cases.

1 Introduction

Many applications these days come with a graphical user interface (GUI). Programs with GUIs are hard to develop due to the inversion of the program's control flow that is common in such programs [9]. UI designers often envision user interactions followed by programmers writing event handlers that describe the way in which a program should respond to certain events. It is hard to predict when event handlers are executed as this depends on the completion time of previously started computation threads and user behaviour. Also it is common for a developer to forget some sequences of events. While there exist many UI design tools, the glue code implementation for UI functionality is normally still done manually. Thus, GUI code development is error-prone and requires many iterations.

Model-based development of user interactions can mitigate this problem, but is not commonly done in industry yet. For general modelling tasks of interactive systems, formal methods are often considered too complex to learn and too inflexible to perform tasks such as rapid prototyping [7]. Part of the reason for this judgement is the necessity to write additional models, which makes it difficult to justify the additional development time. This observation motivated us to research the automatic GUI glue code synthesis from specifications that

* This work was supported by the German Science Foundation (DFG) under Grant No. 322591867.

encode *requirements* rather than manually designed (and modelled) interaction schemes. As detailed by [19], the use of formal methods forces software engineers to program more simply and clearly, thus preventing many software faults. Furthermore, our approach has the potential to actually speed up the development of GUI glue code.

Synthesis of GUI glue code from formal specifications is a special case of reactive synthesis [3], where a system that continuously interacts with its environment is computed that satisfies its specification for all possible input sequences. Linear temporal logic (LTL, [18]) is a commonly used specification language in this context as it allows to reason over chains of events that usually occur during execution of a system. Traditional reactive synthesis frameworks have a few limitations that prevent them from being applied to GUI glue code synthesis:

1. In synthesis from LTL, the system is assumed to read input bit values and set output bit values in every time step. This is not the case with user interfaces, where the controller responds to every input event by executing a sequence of actions which cannot happen in parallel. There can also be a last input event after which the system and the controller stall.
2. Specifications for user interfaces are normally huge, requiring a very high scalability of the synthesis approach, which traditional approaches for synthesis from LTL specifications do not provide.
3. Synthesis approaches that trade the full expressivity of LTL against improved efficiency, such as Generalized Reactivity(1) Synthesis [4], cannot deal with specifications parts that describe chains of events, which are common in user interface specifications.
4. The application-specific quality metrics for UI glue code, such as starting computation as quickly as possible and enabling UI elements such as buttons whenever possible cannot be accurately captured by traditional quality metrics in reactive synthesis, such as maximising pay-offs in games [2].

Despite these limitations, reactive synthesis can still be applied to GUI glue code synthesis if we can carefully craft the approach in terms of specification language, its semantics, and the game solving approach used. We developed one such approach, which we draft in the next sections.

2 State of the Art

Writing correct user interface code is difficult [13]. The greatest challenge is that the UI should be well suitable and usable by wide range of user groups. When user interfaces are redesigned multiple times in between phases of usability testing, current techniques help little to avoid concurrency bugs when altering code. This challenge has been partially addressed in the work on multi-modal systems and various forms of model-based interfaces ([6]; [14]). Although testing can be applied in the latter case, it is particularly challenging in the GUI case [10], as it never guarantees the absence of bugs in the code. Formal verification can be applied as well. For instance, a modern approach by Masci et al. [12]

offers correctness checking and rapid prototyping of user interfaces in the early development stages. For their approach, the model still needs to be manually written, and it has to be maintained to be useful for post-deployment changes to the software, which takes a lot of effort. Similarly, a generic approach to verify interactive systems has been proposed in [16], where each system component is described as a module that interacts through channels. This approach allows the *plasticity* of UI's to be analyzed. *Plasticity* is the capacity of a UI to withstand variations in its context of use (environment, user, platform) while preserving usability [20]. The approach used LNT [5], a formal specification language derived from the *Enhanced Language of Temporal Ordering Specifications* (ELOTOS) standard [1]. LOTOS and LNT are equivalent with respect to expressiveness, but have a different syntax. In [17], the authors point out how difficult it is to model a system using LOTOS, when quite simple UI behaviours can easily generate complex LOTOS expressions. The use of LNT alleviates this difficulty. The approach can cover aspects of the users, the user interfaces, and the functional core of the system but not the state of the system. This is the downside of formal verification, where major effort is spent on building a model and only afterwards, bugs in the implementation (or its model) can be found. The concept of synthesis addresses this problem by generating correct-by-construction designs directly from their specifications.

GUI glue code can be formalized as a finite-state machine that can continuously react to user inputs. Synthesizing such code hence fits the concept of reactive synthesis [3], in which such finite-state machines are computed from specifications. Linear temporal logic (LTL) is commonly used as specification language in this context. Despite of the fact that synthesis from LTL has a provably high computational complexity, practical reactive synthesis has seen a substantial improvement in the last few years, with tools such as Strix [15] winning the 2018 Reactive Synthesis Competition for LTL specifications.

3 Our Approach to GUI Glue Code Synthesis

We recently published a novel approach to the synthesis of GUI glue code that combines efficient reasoning with being adapted to GUI glue code [8].

We started with the observation that all events in GUIs, such as UI element clicks and computation threads starting and terminating, are ordered and no two events can happen at the same time. After the GUI glue code hands back control to the UI framework, the system is also blocked from executing further actions before an external event occurs. This is quite different from classical reactive synthesis, where in every step, the input and output atomic propositions have values. We defined a new controller execution semantics for synthesis that captures the particularities of this application domain.

Then, we identified a class of automata to which the usual specification parts for UI applications can be translated and that permits an efficient synthesis approach. These so-called *universal very-weak automata* have been found to be interesting from a theoretical point of view, as they capture exactly the temporal properties that can be expressed both in the temporal logics LTL and ACTL [11].

We developed an approach to reduce the synthesis problem from specifications given as universal very weak automata to solving a game between two players. The approach reduces size of the game based on exploiting the special execution semantics that GUI glue code has. The resulting games are small enough to be efficiently solved.

We implemented the overall approach in a prototype tool that takes a formal specification for GUI glue code, performs the synthesis approach outlined above, and finally emits GUI glue code in the programming language `Java` that can be compiled into an Android application.

4 Presentation Overview

During the proposed presentation at the FMIS workshop, we will focus on explaining the main ideas of synthesizing GUI glue code, i.e., where the differences to classical model-based development of UI controllers are, how specifications for such GUI glue code look like, and what it means to leave the synthesis of such code *fully* to the synthesis tool, without building a model of the interaction between the system and the user by hand. An example Android application (for cost splitting between members of a team) will serve as our running example.

The aim of the talk is to stimulate a discussion on whether the idea to synthesize GUI code rather than modelling the desired behavior by hand is a promising path to correct-by-construction UI interaction, and how it can be integrated with more wholistic approaches to assure the correctness of systems with a UI.

References

1. IEC ISO (2001) enhancements to LOTOS (E-LOTOS). international standard 15437
2. Bloem, R., Chatterjee, K., Henzinger, T.A., Jobstmann, B.: Better quality in synthesis through quantitative objectives. In: Bouajjani, A., Maler, O. (eds.) CAV. LNCS, vol. 5643, pp. 140–156. Springer (2009)
3. Bloem, R., Chatterjee, K., Jobstmann, B.: Graph games and reactive synthesis. In: et al., E.M.C. (ed.) Handbook of Model Checking, pp. 921–962. Springer (2018)
4. Bloem, R., Jobstmann, B., Piterman, N., Pnueli, A., Sa'ar, Y.: Synthesis of reactive(1) designs. J. Comput. Syst. Sci. **78**(3), 911–938 (2012)
5. Champelovier, D., Clerc, X., Garavel, H., Guerte, Y., Lang, F., Serwe, W., Smeding, G.: Reference manual of the LOTOS NT to LOTOS translator (version 5.0). INRIA/VASY (2010)
6. Coutaz, J.: User interface plasticity: Model driven engineering to the limit! In: Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems. pp. 1–8. ACM (2010)
7. Dix, A.J.: Formal methods for interactive systems, vol. 16. Academic Press London (1991)

8. Ehlers, R., Adabala, K.: Reactive synthesis of graphical user interface glue code. In: Accepted at the Automated Technology for Verification and Analysis - 17th International Symposium (ATVA) (2019), preprint available at `https://www.ruediger-ehlers.de/publications.html`.
9. Fayad, M.E., Schmidt, D.C.: Object-oriented application frameworks - introduction. Commununications of the ACM **40**(10), 32–38 (1997)
10. Hellmann, T.D., Hosseini-Khayat, A., Maurer, F.: Agile interaction design and test-driven development of user interfaces – a literature review. In: Dingsøyr, T., Dybå, T., Moe, N.B. (eds.) Agile Software Development, pp. 185–201. Springer Berlin Heidelberg (2010)
11. Maidl, M.: The common fragment of CTL and LTL. In: FOCS. pp. 643–652 (2000)
12. Masci, P., Oladimeji, P., Zhang, Y., Jones, P.L., Curzon, P., Thimbleby, H.W.: Pvsio-web 2.0: Joining PVS to HCI. In: 27^{th} International Conference on Computer Aided Verification (CAV). pp. 470–478 (2015)
13. Masci, P., Zhang, Y., Jones, P.L., Curzon, P., Thimbleby, H.W.: Formal verification of medical device user interfaces using PVS. In: Gnesi, S., Rensink, A. (eds.) FASE. LNCS, vol. 8411, pp. 200–214. Springer (2014)
14. Meixner, G., Paternò, F., Vanderdonckt, J.: Past, present, and future of model-based user interface development. i-com Zeitschrift für interaktive und kooperative Medien **10**(3), 2–11 (2011)
15. Meyer, P.J., Sickert, S., Luttenberger, M.: Strix: Explicit reactive synthesis strikes back! In: Chockler, H., Weissenbacher, G. (eds.) CAV. LNCS, vol. 10981, pp. 578–586. Springer (2018)
16. Oliveira, R., Dupuy-Chessa, S., Calvary, G.: Plasticity of user interfaces: formal verification of consistency. In: Proceedings of the 7th ACM SIGCHI symposium on engineering interactive computing systems. pp. 260–265. ACM (2015)
17. Paterno', F.: Formal reasoning about dialogue properties with automatic support. Interacting with computers **9**(2), 173–196 (1997)
18. Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977. pp. 46–57. IEEE Computer Society (1977)
19. Stavely, A.M.: Toward zero-defect programming. Addison-Wesley Longman Publishing Co., Inc. (1998)
20. Thevenin, D., Coutaz, J.: Plasticity of user interfaces: Framework and research agenda. In: Interact. vol. 99, pp. 110–117 (1999)

Formal Modelling of Safety-Critical Interactive Devices using Coloured Petri Nets

Sapna Jaidka[1], Steve Reeves[2], and Judy Bowen[3]

[1] University of Waikato, Hamilton, New Zealand
sapnajaidka87@gmail.com
[2] University of Waikato, Hamilton, New Zealand
stever@waikato.ac.nz
[3] University of Waikato, Hamilton, New Zealand
judy.bowen@waikato.ac.nz

Abstract. Formal modelling is now widely applied for creating models of safety-critical interactive systems. Most approaches built so far either focus on the user interface or on the functional part of a safety-critical interactive system. This paper aims to apply formal methods for modelling and specifying the user interface, interaction and functional aspects of a safety-critical system in a single model using Coloured Petri Nets (CPN). We have used CPNs because of its expressive graphic representation and the ability to simulate the system behaviour. The technique is illustrated through a case study of the Niki T34 Infusion Pump.

Keywords: Formal Modelling · Formal Method Integration · Coloured Petri Nets.

1 Introduction

Safety should be a central consideration in the development of safety-critical interactive systems. There are many systems that are considered as safety-critical interactive systems where the interaction occurs via a user or, perhaps, via an automatic manufacturing system (production cell) where sensors are interacting among themselves. It is very important to ensure that all these types of system behave correctly because failure can cause significant damage to property, the environment or even human life. Researchers have been working for many years to solve problems or issues in safety-critical systems due to poor user interfaces or functional errors. The use of formal methods for modelling is often recommended as a way of raising confidence in such systems. The focus of this paper is on user interfaces as well as the underlying system functionality of safety-critical interactive systems.

The work here aims to apply formal methods for modelling and specifying the user interface, interaction and functional aspects of the system in a single model. This technique has its starting point in several formal specification techniques: Z, Presentation Interaction Models (PIMs) and Presentation Models (PMs) (as for instance in [5]). From this (existing) basis we create a Coloured Petri Net (CPN) model of a system which will have the required aspects of Z, PMs and PIMs expressed within it. To specify safety-critical systems adequately, all three aspects, behavioural, functional and user interface/interaction must be taken into account, hence our investigation of the combination of models. In summary, our plan here is to show how an existing, accepted way of formally modelling systems via PM/PIM/Z can be re-cast in the single formalism of CPN, and then in future, having shown the CPN models are as expressive as the PM/PIM/Z models, we can move straight from the system to a CPN model of it.

We have chosen CPNs mainly because of the state space analysis-based methods made possible within the CPN Tool, based on support for the state space graph and the strongly connected components graph to be automatically generated. Once we have these then functions can be written in the SML-subset available in CPN Tools which allow many useful further checking and testing mechanisms. Comparing this with other possibilities, there is a tool called RENEW for Reference Nets but it does not generate the state space graph [18]. Also we find in the literature that the Reference Net models or even Object Petri Nets models get transformed into behaviourally equivalent CPNs for adapting CPN analysis techniques, as mentioned in [19].

The motive for doing this move is two-fold: the existing method results in three separate models, and the drawback with this is that a lot of work is required to do the coupling of functional behaviour with interactive elements to ensure consistency [8][26]. Moreover, these models need to be combined in order to verify safety properties which might relate to functional constraints, interface constraints or both. The new technique results in a single model capturing all three aspects and all the connections between them. So, the benefit of having all aspects in a single model is that there is no work required to combine them for analysis.

2 Related Work

In the early years, the main focus of formal methods was on modelling and specifying the functional part of a system. User-interfaces and interaction were not considered important because systems used to be very simple. But now as the interfaces have become more complex, so their design and analysis is very obviously important. There were some formal methods which were used to model and verify both user-interface and interaction, for example, Jacob in [13] has used techniques based on state transition diagrams and BNF to specify the user-interface and Dix and Runciman in [11] focused on creating abstract models for user interfaces and interaction. However, formal methods for system development and those for modelling user interfaces and interaction were considered separate. There exists far less work on the combination of both these aspects.

The Food and Drug Administration (FDA) has been working with academic collaborators to develop model-based engineering methods [1]. Figueiredo et al. [2] presented the main advantages of a formal language that is able to be used in the construction of reference models for the medical devices domain and conducted a case study on a specification of a medical device. Masci et al. also presented a case study on a generic PCA infusion pump in [21] and verified the user-interface using the Prototype Verification System (PVS) and also formally modelled the requirements of the interface in [23]. Campos and Harrison presented a case study on modelling and analysis of the Alaris infusion pump using their IVY tool [9].

Petri nets form a powerful modelling language and higher level nets, like Coloured Petri Nets, are used for modelling critical scenarios like railway systems and other safety-critical systems [3] [14]. There exist formalisms like HAMSTERS [20] that focus on task models, and also ICOs [22] and the APEX framework [24]. The main difference concerning ICOs is around levels of abstraction, because they take an object-oriented view whereas we are committed to more abstract prior models. All these studies either focus on the functional part of the system or on the user-interface and interaction. We present a technique to combine all aspects in a single model.

3 Coloured Petri Nets and their extensions

Coloured Petri Nets (CPN) is a language used for the modelling and validation of hardware and software systems. The existing CPN Tool [16] helps in constructing a model and performing syntax checking. Also simulations can be performed and we can see at every step how the model is behaving. Automatic generation of full and partial state spaces helps in analyzing and verifying the net model.

We will commence with the formal definition of Coloured Petri Nets [16]. The following are assumed to be defined: *EXPR* denotes the set of expressions provided by the inscription language, i.e., CPN ML [15]. Given an expression $e \in \mathit{EXPR}$, the ***type*** of e is represented by $\mathit{Type}[e]$. The set of ***variables*** in an expression e is denoted by $\mathit{Var}[e]$. V denotes the set of (all) variables. By S_{MS}, we denote the set of all *multi-sets* over the set S [15].

Definition 1. *A Coloured Petri Net is a tuple* $(CS, P, T, A, N, C, G, E, I)$ *such that [15]:*

(i) *CS is a finite set of non-empty types, called colour sets.*
(ii) *P is a finite set of places.*
(iii) *T is a finite set of transitions.*
(iv) *A is a finite set of directed arcs such that connect places and transitions.*
(v) *N is a node function. It is defined from A into $P \times T \cup T \times P$ and shows, for each arc, which places and transitions are connected by that arc.*
(vi) *C is a colour function. It is defined from P into CS.*
(vii) *G is a guard function. It is defined from $T \to \mathit{EXPR}$ such that: $\forall\, t \in T : \mathit{Type}[G(t)] = \mathit{Bool} \wedge \mathit{Type}[\mathit{Var}[G(t)]] \subseteq CS$.*
(viii) *E is an arc expression function. It is defined from A into expressions such that: $\forall\, a \in A : \mathit{Type}[E(a)] = C(p(a))_{MS} \wedge \mathit{Type}[\mathit{Var}[E(a)] \subseteq CS]$ where $p(a)$ is a place of $N(a)$.*
(ix) *I is an initialization function. It is defined from P into closed expressions such that $\forall\, p \in P : \mathit{Type}[I(p)] = C(p)_{MS}$.*

Definition 2. *A distribution of tokens on the places is called a **marking**. A marking M is a function that maps each place p into a multi-set of values $M(p)$ representing the marking of p. The **initial marking** is denoted by M_0.*

Definition 3. *The **variables of a transition t** is denoted, by overloading function V, as $\mathit{Var}[t] \subseteq V$ and is defined so it consists of the free variables appearing in the guard of t and in the arc expressions of arcs connected to t. A **binding** of a transition t is a function b that maps each variable $v \in \mathit{Var}[t]$ into a value $b(v) \in \mathit{Type}[v]$. It is extended to expressions from EXPR in the obvious way. The application of binding b to expression e is written $e\langle b\rangle$. The set of all bindings for t is denoted by $B(t)$.*
*A **binding element** is a pair (t, b) where $t \in T$ and $b \in B(t)$.*
We often write an arc expression $E(a)$ as $E(p, t)$ or $E(t, p)$ when $N(a) = (p, t)$ or $N(a) = (t, p)$, respectively, as a suggestive shorthand.

Definition 4. *For a binding element (t,b) to be **enabled** in a marking M there are two conditions to satisfy: firstly, the corresponding guard expression must evaluate to True. Secondly, for each place p, an arc expression $E(p, t)$ has to be evaluated using the binding b so that $E(p, t)\langle b\rangle \leq M(P)$. This means that for each place p there should be enough tokens there of the right form so that transition t can remove the required number of tokens.*

This means that in an enabled binding element (t,b), the multi-set of tokens removed from an input place p when t occurs with a binding b is given by $E(p,t)\langle b\rangle$, and similarly $E(t,p)\langle b\rangle$ is the multi-set of tokens added to an output place p.

Definition 5. *A **step** Y is a non-empty, finite multi-set of binding elements. A step Y is **enabled** in a marking M iff the following property is satisfied [15]: $\forall\, p \in P : \sum_{(t,b)\in Y} E(p,t)\langle b\rangle \ \leq M(p)$.*
When a step Y is enabled in a marking M_1 it may occur, changing the marking M_1 to another marking M_2, defined by: $\forall\, p \in P : M_2(p) = (M_1(p) - \sum_{(t,b)\in Y} E(p,t)\langle b\rangle) + \sum_{(t,b)\in Y} E(t,p)\langle b\rangle$, which is to say that when a step happens, tokens are removed from the starting place of a transition and placed in the ending place of that transition.

3.1 Hierarchical Coloured Petri Nets

Hierarchical Coloured Petri Nets allow models to be divided into modules. This allows the model to be organized into several pages. There are two ways to interconnect these several pages: *substitution transitions and fusion places* [17]. In this paper we are using *fusion places*. *Fusion places* are places which are functionally identical, so they have the same marking.

4 Presentation Model

A *Presentation Model* (PM) [6] describes the existence, category and behaviour of the widgets (interactive elements) of a user interface. Widgets are categorized using the widget categorization hierarchy given in [4]. A presentation model typically consists of several component presentation models which could be understood as the states of the user interface.

Presentation models consist of two parts: declaration and definition.

$$\langle declaration\rangle \ ::= \ WidgetName\{\langle ident\rangle\}^+ \ Category\{\langle ident\rangle\}^+ \ Behaviour\{\langle ident\rangle\}^*$$

The declarations introduce the three sets of identifiers which can be used within the definitions. *WidgetName* is a list of names of widgets. *Category* refers to the description of widget categories. *Behaviour* shows what behaviour a widget has associated with it (and it can be empty). Behaviours are divided into two categories. The first is called a system behaviour (S-behaviour) which refers to the underlying non-interactive system and the second category is called an interactive behaviour (I-behaviour) that represents user interface functionality, which changes things about the user interface itself, like changing screens.

A definition consists of one or more identifiers for presentation models.

$$\langle definition\rangle ::= \{\langle pname\rangle is\langle pexpr\rangle\}^+ \ where\ \langle pname\rangle ::= \langle ident\rangle \ and\ \langle pexpr\rangle ::= \{\langle widgetdescr\rangle\}^+$$

Each state of the system is described in a separate component presentation model by the means of widget descriptions. A widget description consists of a triple, the widget name, the category and the set of behaviours associated with the widget. The syntax of a widget description is as follows:

$$\langle widgetdescr\rangle ::= (\langle widgetname\rangle, \langle category\rangle, (\{\langle behaviour\rangle\}^*))$$

Consider a device as shown in Figure 1 having two buttons *ON* an *OFF* and one *Display*. Pressing the *ON* button will display a start message and the *OFF* button will switch off the device. A PM for the device is given in table 1. The model has three widgets and each widget falls under one of the two categories (*ActCtrl* or *Responder*). *ActCtrl* is shorthand for action control. The simple device's PM has one S-behaviour and two I-behaviours. In table 1, *init, ON, OFF* are three component presentation models. Each component presentation model consists of a set of widget triples. For example, the *ON* component presentation model comprises of three sets of widget triples. The first set of triples means that the widget *Display* is of category *Responder* and has the *S_startmessage* behaviour associated with it. Notice that the behaviour *Quit* is not labelled as an I-behaviour or S-behaviour as it is a special behaviour that terminates both the system and the user interface.

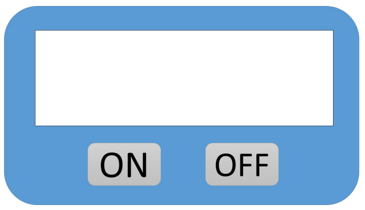


Fig. 1: Simple Device

WidgetName	Display ONButton OFFButton
Category	ActCtrl Responder
Behaviour	S_startmessage I_ON
	I_OFF Quit
Init is	(Display, Responder, ())
	(ONButton, ActCtrl, (I_ON))
	(OFFButton, ActCtrl, (I_OFF))
ON is	(Display, Responder, (S_startmessage))
	(ONButton, ActCtrl, ())
	(OFFButton, ActCtrl, (I_OFF))
OFF is	(Display, Responder, ())
	(ONButton, ActCtrl, (I_ON))
	(OFFButton, ActCtrl, (Quit))

Table 1: Presentation Model of a simple device

4.1 Expressing Presentation Models in CPN

We will first look at the declaration part of a PM which introduces the three sets of identifiers: *WidgetName*, *Category* and *Behaviour*. These three sets of identifiers can be modelled in CPN by:

$$\begin{aligned} colset\ WidgetName &= with\ wid_1 \mid wid_2 \mid ... \mid wid_n; \\ colset\ Category &= with\ cid_1 \mid cid_2 \mid ... \mid cid_n; \\ colset\ Behaviour &= with\ bid_1 \mid bid_2 \mid ... \mid bid_n; \\ colset\ Behaviours &= list\ Behaviour; \end{aligned}$$

where wid_s are the names of the widgets, cid_s are the names of the category of the widgets and bid_s are the names of the S-behaviours and I-behaviours associated with the widgets.

Now we look at the definition part of the presentation model [6]. A widget description which we call as *widgetdescr* is described as a tuple consisting of the widget name, the category and the list of behaviours associated with the widget. A widget description can be written in CPN as:

$$colset\ \ widgetdescr = product\ \ WidgetName\ \ *\ \ Category\ \ *\ Behaviours;$$

Consider again the example shown in Figure 1 and Table 1. The model of this presentation model in CPN is shown in Tables 2 and 3. Each state of the system is described in a separate *component presentation model* by the means of *widgetdescr*. This can be written in CPN as:

$$colset\ pmodel = list\ widgetdescr;$$

colset	WidgetName	=	with Display | ONButton | OFFButton;
colset	Category	=	with ActCtrl | Responder;
colset	Behaviour	=	with S_startmessage | I_ON | I_OFF | Quit;
colset	Behaviours	=	list Behaviour;

Table 2: Presentation Model Declarations of the Simple Device in CPN

val	Init	=	[(Display, Responder, [ ]), (ONButton, ActCtrl, [I_ON]), (OFFButton, ActCtrl, [I_OFF])];
val	ON	=	[(Display, Responder, [S_startmessage]), (ONButton, ActCtrl, [ ]), (OFFButton, ActCtrl, [I_OFF])];
val	OFF	=	[(Display, Responder, [ ]), (ONButton, ActCtrl, [I_ON]), (OFFButton, ActCtrl, [Quit])];

Table 3: Presentation Model Definition of the Simple Device in CPN

To define the component presentation model we will use the value declaration feature of CPN. A value declaration binds a value to an identifier. The component presentation models for the simple device of Figure 1 are given in CPN in Table 3. We can now see what the states of the device are and what widgets are available to the user in every state and what the behaviours of those widgets are. But to understand the navigational possibilities, it is always better to have some graphical representation. Another model, i.e., a presentation and interaction model (PIM), is used for this purpose.

5 Presentation Interaction Models

A presentation and interaction model (PIM) describes the transitions between states [6]. A PIM is the combination of a presentation model and a finite state machine (FSM). A PIM gives a formal meaning to I-behaviours given in the presentation model. The PIM is derived by creating a single state for each of the component presentation models and creating transitions between states based on the relevant I-behaviours, so transitions give the meaning of I-behaviours.

The PIM for the simple device in Figure 1 is given in Figure 2. There are three states: *Init*, *ON* and *OFF*. Figure 1 show what states can be reached from a current state via those I-behaviours.

We now look at a way of expressing all this within CPN. The number of pages in a CPN model of a PIM is the same as the number of component presentation models in the PIM. For example, the simple device PIM shown in Figure 1 has three component presentation models, so there are three pages in the CPN model of this device as shown in figure 3. These pages represent the individual component presentation models. The names of the places are exactly the names of the component presentation models. For the simple device, there are three places: *Init*, *ON* and *OFF*. These places have fusion tags (in blue) named the same as the corresponding component presentation models as shown in figure 3. The component presentation model *Init* has two I-behaviours, which means

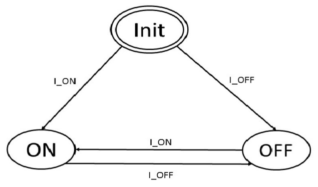


Fig. 2: PIM of Simple Device

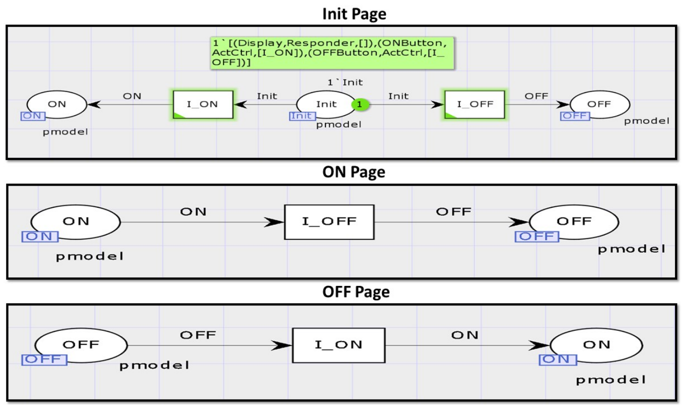


Fig. 3: CPN model of PIM of Simple Device

that from the place *Init* a user can go to two other states *ON* and *OFF*. The page *Init* will have three places: *Init*, *ON* and *OFF* where *Init* is the current state and *ON* and *OFF* are the states a user can go to from *Init*. Every place in the model will be of one type, i.e. *pmodel*. The transitions give formal meaning to the I-behaviours in the presentation model and have the same name as the respective I-behaviours in the presentation model.

5.1 Formal Definition of CPN for modelling PM/PIM

In this section we formalize the definition of how we represent the combination of presentation models and presentation interaction models.

Definition 6. *A non-hierarchical Coloured Petri Net for modelling a PM/PIM combination is a tuple* $(PM, K, P, T, I, CS, A, C, G, E)$ *such that:*

(i) *PM is a finite set of colour sets for representing presentation model declarations from the PM, such that:* $PM = \{WidgetName, Category, Behaviour, Behaviours, widgetdescr, pmodel\}$, *where*
  - *colset* $WidgetName = with\ wid_1 \mid wid_2 \mid ... \mid wid_n$*;*
    - • *where* wid_s *are the names of the widgets in the various component presentation models in PM.*
  - *colset* $Category = with\ cid_1 \mid cid_2 \mid ... \mid cid_n$*;*
    - • *where* cid_s *are the names of the category of the widgets in the various component presentation models in PM.*
  - *colset* $Behaviour = with\ bid_1 \mid bid_2 \mid ... \mid bid_n$*;*
    - • *where* bid_s *are the names of the S-behaviours and I-behaviours associated with the widgets.*
  - *colset* $Behaviours = list\ Behaviour$*;*
  - *colset* $widgetdescr = product\ WidgetName * Category * Behaviours$*;*
  - *colset* $pmodel = list\ widgetdescr$*;*

(ii) *K is a finite set of constants that represents the component presentation models by their names and is such that* $Type[K] = pmodel$.

(iii) *P is a finite set of places, the same size as K, representing the component presentation models where the names of the places and names of the constants are same.*

(iv) *T is a finite set of transitions representing the I-behaviours of the PIM.*

(v) *I is an initialization function that assigns an initial marking to each place. The initialization function $I : P \rightarrow EXPR$ assigns an initialization expression $I(p)$ to each place p such that: $I(p) = k \in K$, i.e., $I(p)$ can be a constant, k, representing a component presentation model.*

(vi) *CS is a finite set of non-empty types, called colour sets with $PM \subseteq CS$.*

(vii) *A is a finite set of arcs as given in definition 1.*

(viii) *C is a colour function. It is defined from P into CS as given in definition 1.*

(ix) *G is a guard function as given in definition 1.*

(x) *E is an arc expression function such that:*
- *For an arc $(p, t) \in A$, connecting a place $p \in P$ and a transition $t \in T$, it is required that the arc expression $E(p, t)$ is the constant $k \in K$ which represents the component presentation model of the place p.*

Having modelled the PM/PIM combination in CPN, we now move to modelling the functionality.

6 Z

Z is a formal specification language which is used to specify and model systems. Z specifications can be recognized by the use of the schema. More detailed information can be found in [25] [10]. Z operation schemas are used to give formal meaning to the S-behaviours of a presentation model. A Z specification for the simple device in Figure 1 is as follows:

This definition introduces a type *MESSAGE* that contains (only) a value *InitializingDevice*.

$MESSAGE ::= InitializingDevice$

The schema *SimpleDevice* is the state space of the model. It says that in each state in the state space there is one observation *display* which can have a value of type *MESSAGE*. In its initial state, the value of the *display* is set to *InitializingDevice*. The schema *Startmessage* refers to the *S_startmessage* behaviour of the presentation model.

SimpleDevice

$display : MESSAGE$

Init

$SimpleDevice$

$display = InitializingDevice$

Startmessage

$\Xi SimpleDevice$
$display! : MESSAGE$

$display! = InitializingDevice$

Now we have a Z specification that gives meaning to the S-behaviours in the presentation model. As we will now see, instead of having three separate models, we can actually include the S-behaviours (functionality) in the CPN model by expressing a Z in CPN and have a single model.

6.1 Expressing Z in CPN

In this section we will explain how the kinds of Z specification [10] used in the existing PM/PIM/Z models can be expressed in CPN using colour sets, an initial expression and arc inscriptions. Notice that we give rules for the small subset of Z which is adequate for our purposes. We do not need all of Z to be modelled.
Cartesian Product: It is a type consisting of ordered pairs. We can use the product colour set of CPN to represent such Z types and can be written in CPN as:

$$colset\ \langle z\_type\_name\rangle = product\ \langle colset\_name_1\rangle * \langle colset\_name_2\rangle * ... * \langle colset\_name_n\rangle;$$

where $colset\_name_1 ... colset\_name_n$ are already defined colour sets which represent types in Z.
Built-in type: Z provides a single built-in type, namely the type of integers $\mathbb{Z}$. We can write this in CPN using the integer colour set. So the declarations

$$colset\ INT = int; \quad var\ n : INT;$$

will create a colour set INT which defines INT as integers, and a variable n such that the value of n is an integer.
Power sets: The power set operator $\mathbb{P}$ (giving "the set of all subsets" of a set) is an elementary type constructor often used in Z. If we want to write such a type in CPN, then in this work we have decided that the list colour set is used[4]. The syntax for writing power sets of Z based on this decision in CPN is:

$$colset\ \langle z\_type\_name\rangle = list\ \langle colset\_name\rangle;$$

Axiomatic Definition: If we have an axiomatic definition:

$$\begin{array}{|l} hours : \mathbb{P}\,\mathbb{N} \\ \hline hours = 0\mathinner{..}24 \end{array}$$

This can be written in CPN as: $colset\ hours = int\ with\ 0..24;$
Z schemas: Z schemas are used to specify the state space and operations of the system. To write the declaration part of a Z schema in CPN, we use the record colour set:

$$colset\langle Z\rangle = record\ id_1 : type_1 * ... * id_n : type_n;$$

where $id_1..id_n$ are Z observations and $type_1..type_n$ are their corresponding types (which are already declared colour sets using the rules as explained above) as they appear in the declarations of the schema we are modelling. The initialization schema of a Z specification is represented as an initial marking in Coloured Petri Nets. The predicate part of a schema will give us expressions on arcs of certain transitions, as we will see later.

[4] We model only systems with finite components, so modelling the power set with lists is no restriction on our expressiveness.

6.2 Formal Definition of CPN for Modelling PM/PIM/Z

Definition 7. *A non-hierarchical Coloured Petri Net for modelling a PM/PIM/Z model is a tuple* $(Z, PM, K, P, T, I, CS, A, C, G, E)$ *such that:*

(i) *Z is a finite set of colour sets representing the Z state schema of the original model with declarations* $id_1 : type_1 ... id_n : type_n$ *, such that:* $Z = \{type_1, type_2, ..type_n, Z\}$*, where colset* $type_1$ *;.....colset* $type_n$*; colset* $Z = record\ id_1 : type1 * ... * id_n : type_n$*;*
(ii) *PM is a finite set of colour sets representing PM declarations as given in definition 6.*
(iii) *K is a finite set of constants representing component presentation models as in definition 6.*
(iv) *P is a finite set of places such that* $\mid P \mid = \mid K \mid +1$*. The constant k in the set K can be mapped to the place p in the set of P with the same name. There is an additional place named Z that represents the state schema.*
(v) *T is a finite set of transitions representing both I-behaviours and S-behaviours.*
(vi) *I is an initialization function that assigns an initial marking to each place.*
(vii) *CS is a finite set of non-empty types, called colour sets, with* $PM \subseteq CS$ *and* $Z \subseteq CS$*.*
(viii) *A is a finite set of arcs as given in definition 6.*
(ix) *C is a colour function as given in definition 6.*
(x) *G is a guard function as given in definition 6.*
(xi) *E is an arc expression function such that:*

- *For an arc* $(p, t) \in A$*, where t is an I-behaviour transition, it is required that the arc expression* $E(p, t)$ *is the name* $k \in K$ *which represents the component presentation model. Similarly for a directed arc from a transition representing an I-behaviour to a place.*
- *For an arc* $(p, t) \in A$ *where p represents a state in the Z state space, and a transition* $t \in T$*, representing an S-behaviour, it is required that the arc expression* $E(p, t)$ *assigns each observation name appearing in S to a new, unique variable (say s).*

7 Case Study: NIKI T34 Infusion Pump

The T34 is a compact and lightweight syringe pump used to deliver drugs. Figure 4 shows an image of the pump. More information about the pump can be found in [12]. There are ten widgets in the T34 infusion pump as shown in Figure 4 : *LeftFFSK, RightBackSK, OnOffButton, UpPlusSK, DownMinusSK, Display, NoStopSK, InfoSK, YesStartSK* and *Timeout*.

There is a total of nineteen states, one of which the pump can be in at any given point in time which are: *LoadSyringe, Init, BatteryLevel, SetVolume, SetDuration, RateSet, RateConfirm, ConfirmSettings, StartInfusingConfirm, Infusing, Paused, Inittwo, Resume, InfusionStatus, BatteryStatus, EventLog, ChangeSetUp* and *TimeOut*. We can, typically, get to know this by actually experimenting with the device. We might also read the user manuals, but this is not recommended since user manuals are, worryingly, notoriously unreliable [7].

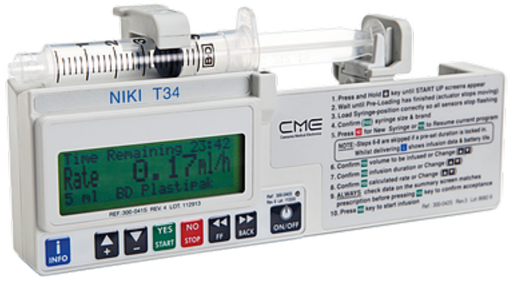


Fig. 4: Niki T34 Syringe Pump

7.1 Modelling T34 pump in CPN

The declaration part of the presentation model is given in Table 4. *WidgetName* is an enumeration type and represents the names of the widgets of the pump. *Category* is an enumeration type that describes the categories of the widgets. *Behaviour* is an enumeration type that represents a set of all behaviours. Because of space restrictions only a few of the full set of behaviours are given here[5]. *Behaviours* is of type list, so a widget can have more than one behaviour. *Behaviours* is a list of behaviours where the names of the behaviours is taken from the *Behaviour* colour set. *widgetdescr* is of type product. It represents a triple (*WidgetName*, *Category*, [*Behaviours*]). *pmodel* is a list colour set and represents the component presentation model. Now we look at the definition part of the presentation model for the Niki T34 infusion pump. As there are nineteen states, the number of component presentation models will be the same. We give component presentation models for *LoadSyringe* and *Info* as representative of those for the whole T34 in Table 5.

```
colset WidgetName =  with LeftFFSK | RightBackSK | OnOffButton | UpPlusSK | DownMinusSK |
                     Display | NoStopSK | InfoSK | Timeout | YesStartSK;
colset Category =    with ActionControl | MValResponder | System | display;
colset Behaviour =   with S_SyringeWarnings | S_MoveActuatorFwd | S_ArmWarning |
                     I_Init | S_SyringeDisplay | S_ScrollSyringeList | I_SetVolume |...;
colset Behaviours =  list Behaviour;
colset widgetdescr = product WidgetName * Category * Behaviours;
colset pmodel =      list widgetdescr;
```

Table 4: Modelling of the presentation model declarations

```
val LoadSyringe = [(Display, MValResponder, [S_SyringeWarnings]),
(InfoSK, ActionControl, [I_Info]), (UpPlusSK, ActionControl, []),
(DownMinusSK, ActionControl, []), (YesStartSK, ActionControl,[]),
(NoStopSK, ActionControl, []),
(LeftFFSK, ActionControl, [S_MoveActuatorFwd,S_ArmWarning]),
(RightBackSK, ActionControl, [S_MoveActuatorBwd, S_ArmWarning]),
(OnOffButton, ActionControl, [Quit]), (Timeout, System, [I_Init])];

val Info = [(Display, MValResponder, [S_InfoList]),
(InfoSK, ActionControl, [S_KeypadLock]),
(UpPlusSK, ActionControl, [S_ScrollInfoListUp]),
(DownMinusSK, ActionControl, [S_ScrollInfoListDown]),
(YesStartSK, ActionControl, [I_BatteryLevel,I_Init,I_RateSet,I_EventLog,I_ChangeSetUp]),
(NoStopSK, ActionControl, [I_Init]), (LeftFFSK, ActionControl, [ ]),
(RightBackSK, ActionControl, [ ]), (OnOffButton, ActionControl, [Quit, I_Init]),
(Timeout, System, [ ])];
```

Table 5: CPN version of the PMs

[5] See https://github.com/sapnajaidka/NikiT34-CPN-Model for complete details

```
colset YesNo = with yes | no;
colset SyringeBrand = with BDPlastipak;
colset PerCent = int with 0..100;
colset millilitres= int with 0..100;
colset millimeters = int with 0..10;
colset hours = int with 0..24;
colset minutes = int with 0..59;
colset millilitresperhour = int with 0..100;
colset Z = record BC:PerCent * KP:YesNo * PL:YesNo * TL:YesNo * BD: SyringeBrand *
SS: millilitres * VL: millilitres * PP : millimeters * SOK: YesNo * BCPOK: YesNo *
SR: YesNo * VTBI: millilitres * HH: hours * MM: minutes * IR: millilitresperhour;
```

Table 6: Colour sets and variables for T34 Pump

The complete Z specification for the T34 pump can be found in[6]. The Z types for the T34 pump expressed in CPN are shown in table 6. *YesNo* is declared as the enumerated colour set that can have exactly two values *yes* or *no*. *SyringeBrand* is declared as enumerated colour set with one value *BDPlastipak*. *PerCent*, *millilitres*, *millimeters*, *hours* and *minutes* are declared as integer colour sets. *Z* is a record colour set with a record of all the observations of the state schema with their corresponding types. It represents the *T*34 state schema[7]. As the Z operation schemas would be expressed as arc inscriptions so we need to declare variables which could be bound to different values of their respective colour sets during simulation. There are fifteen variables *BC*, *KP*, *PL*, *TL*, *BD*, *SS*, *VL*, *PP*, *SOK*, *BCPOK*, *SR*, *VTBI*, *HH*, *MM* and *IR*.

There are nineteen component presentation models for the Niki T34 infusion pump, therefore, the CPN model of the pump has nineteen pages which are interconnected by fusion places. Using the CPN Tool, we now create a model that shows all the states with I-behaviours (which show navigational possibilities representing interactivity) and S-behaviours (which represent underlying system functionality). The resulting CPN model has all the important aspects (functional, user-interface and interaction) of the original PM/PIM/Z expressed within it.

The structure of the *LoadSyringe* page is shown in Figure 5. The three places: *LoadSyringe*, *Init* and *Info* represent the states of the system. The marking on the place *LoadSyringe* shows the definition of the *LoadSyringe* component presentation model which gives information about the available widgets. The marking on the *LoadSyringe* page also shows that there are four S-behaviours: *S_SyringeWarnings* and *S_ArmWarning* display warning messages on the screen and *S_MoveActuatorBwd and S_MoveActuatorFwd* are the functions that move the syringe plunger forward and backward. The Z specification[8] for these S-behaviours are not modelled here to keep the size of the state space small, so just the fusion place *Z* is added to the page and represents only the Z *Init* schema.

[6] https://github.com/sapnajaidka/Niki-T34-Z-specification

[7] To make the description short and easy to read we have used abbreviated names for the Z observations. In this declaration *BC* stands for *BatteryCharge*, *KP* is for *KeyPadLocked*, *PL* is for *ProgramLocked*, *TL* is for *TechMenuLocked*, *BD* is for *Brand*, *SS* is for *SyringeSize*, *VL* is for *VolumeLeft*, *PP* is for *PlungerPosition*, *SOK* is for *SyringeOK*, *BCPOK* is for *BarrelOK*, *CollarOK*, *PlungerOK*, *SR* is for *SystemReady*, *HH* is for *Hours*, *MM* is for *Minutes* and *IR* is for *InfusionRate*

[8] See https://github.com/sapnajaidka/Niki-T34-Z-specification

The component presentation model for the *LoadSyringe* has two I-behaviours: *I_Init* and *I_Info* as shown in Table 3 and as shown by the initial marking on that place, so there are two transitions, namely *I_Init* and *I_Info*. Figure 5 clearly shows that from the *LoadSyringe* state, the user can go to either the *Init* state or *Info* state by firing the transitions *I_Init* or *I_Info*.

In Figure 5, on the arc going into place/state *Init*, we have the expression *Init* which tells us the relevant presentation model for this state and on the arc going into place *Info*, we have the expression *Info* which tells us the relevant presentation model for the state *Info*. If the user asks for information, i.e., if *I_Info* transition is fired, then the user goes to the *Info* state. Figure 6 shows the

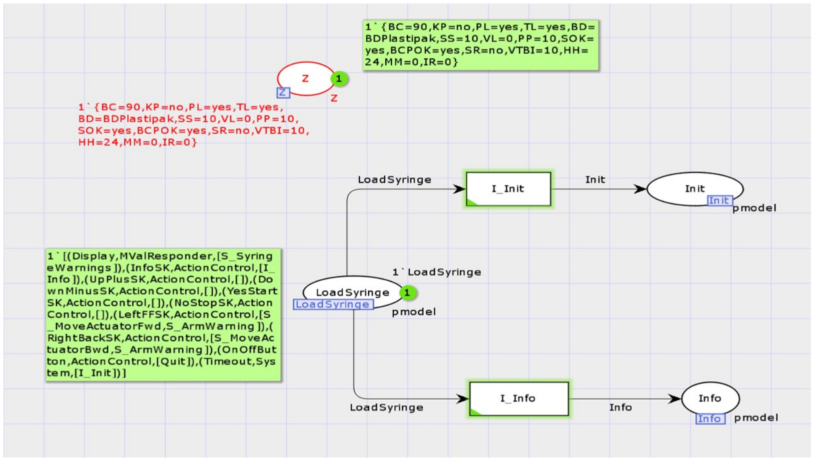

1`{BC=90,KP=no,PL=yes,TL=yes,BD=BDPlastipak,SS=10,VL=0,PP=10,SOK=yes,BCPOK=yes,SR=no,VTBI=10,HH=24,MM=0,IR=0}
Z
1
Z
Z
1`{BC=90,KP=no,PL=yes,TL=yes,BD=BDPlastipak,SS=10,VL=0,PP=10,SOK=yes,BCPOK=yes,SR=no,VTBI=10,HH=24,MM=0,IR=0}
LoadSyringe
I_Init
Init
Init
Init
pmodel
1`[(Display,MValResponder,[S_SyringeWarnings]),(InfoSK,ActionControl,[I_Info]),(UpPlusSK,ActionControl,[]),(DownMinusSK,ActionControl,[]),(YesStartSK,ActionControl,[]),(NoStopSK,ActionControl,[]),(LeftFFSK,ActionControl,[S_MoveActuatorFwd,S_ArmWarning]),(RightBackSK,ActionControl,[S_MoveActuatorBwd,S_ArmWarning]),(OnOffButton,ActionControl,[Quit]),(Timeout,System,[I_Init])]
1`LoadSyringe
LoadSyringe
1
LoadSyringe
pmodel
LoadSyringe
I_Info
Info
Info
Info
pmodel

Fig. 5: LoadSyringe Page with Z

structure of the *Info* page. The *Info* state shows a list of options that a user can select to see status and change settings. There are seven places in the *Info* page: *Info*, *BatteryLevel*, *Init*, *ChangeSetUp*, *Eventlog*, *RateSet* and *Z*. The marking on the place *Info* is a token showing the definition of a component presentation model *Info* which provides information about the available widgets and tells us which button press results in what state. The marking shows that there are five I-behaviours and four S-behaviours. As we are not modelling the S-behaviours which just display messages on the screen, the page *Info* has just one S-behaviour transition *S_KeyPadLocked*. There are six transitions: *I_BatteryLevel*, *I_ChangeSetUp*, *I_Init*, *I_EventLog*, *I_RateSet* and *S_KeyPadLocked*. These transitions give meaning to the I-behaviours and S-behaviours given for the definition of the component presentation model *Info*.

If the user gives a long press to the *Info* button on the device, the keypad gets locked or unlocked. This behaviour changes the value of the observation *KP* from *no* to *yes* and vice-versa. In the CPN model, the transition *S_KeyPadLocked* represents this behaviour. There are two arcs needed to model this (going to and from the place *Z*). The arc from *Z* to *S_KeyPadLocked* simply contains assignments which set each variable to its current value (where the variables are the ones that model the observations from the Z operation schema *KeyPadLocked* where they will appear on the left of each equation in primed form). This set of assignments "picks up" the current values of the variables ready to be used by the second arc. This second arc, the one from *S_KeyPadLocked* to

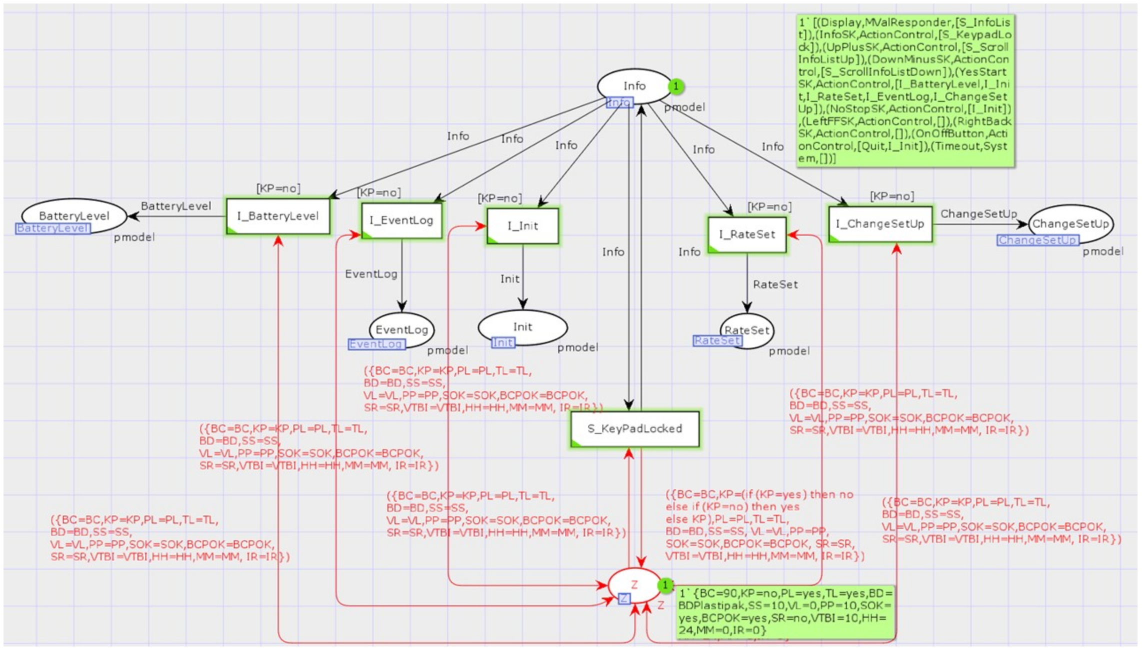

1`[(Display,MValResponder,[S_InfoList]),(InfoSK,ActionControl,[S_KeypadLock]),(UpPlusSK,ActionControl,[S_ScrollInfoListUp]),(DownMinusSK,ActionControl,[S_ScrollInfoListDown]),(YesStartSK,ActionControl,[I_BatteryLevel,I_Init,I_RateSet,I_EventLog,I_ChangeSetUp]),(NoStopSK,ActionControl,[I_Init]),(LeftFFSK,ActionControl,[]),(RightBackSK,ActionControl,[]),(OnOffButton,ActionControl,[Quit,I_Init]),(Timeout,System,[])]
Info
1
Info
pmodel
Info
Info
Info
Info
Info
[KP=no]
[KP=no]
[KP=no]
[KP=no]
[KP=no]
BatteryLevel
BatteryLevel
BatteryLevel
pmodel
I_BatteryLevel
I_EventLog
I_Init
I_RateSet
I_ChangeSetUp
ChangeSetUp
ChangeSetUp
ChangeSetUp
pmodel
Info
Info
EventLog
Init
RateSet
EventLog
EventLog
pmodel
Init
Init
pmodel
RateSet
RateSet
pmodel
({BC=BC,KP=KP,PL=PL,TL=TL, BD=BD,SS=SS, VL=VL,PP=PP,SOK=SOK,BCPOK=BCPOK, SR=SR,VTBI=VTBI,HH=HH,MM=MM, IR=IR})
S_KeyPadLocked
({BC=BC,KP=KP,PL=PL,TL=TL, BD=BD,SS=SS, VL=VL,PP=PP,SOK=SOK,BCPOK=BCPOK, SR=SR,VTBI=VTBI,HH=HH,MM=MM, IR=IR})
({BC=BC,KP=KP,PL=PL,TL=TL, BD=BD,SS=SS, VL=VL,PP=PP,SOK=SOK,BCPOK=BCPOK, SR=SR,VTBI=VTBI,HH=HH,MM=MM, IR=IR})
({BC=BC,KP=KP,PL=PL,TL=TL, BD=BD,SS=SS, VL=VL,PP=PP,SOK=SOK,BCPOK=BCPOK, SR=SR,VTBI=VTBI,HH=HH,MM=MM, IR=IR})
({BC=BC,KP=(if (KP=yes) then no else if (KP=no) then yes else KP),PL=PL,TL=TL, BD=BD,SS=SS, VL=VL,PP=PP, SOK=SOK,BCPOK=BCPOK, SR=SR, VTBI=VTBI,HH=HH,MM=MM, IR=IR})
({BC=BC,KP=KP,PL=PL,TL=TL, BD=BD,SS=SS, VL=VL,PP=PP,SOK=SOK,BCPOK=BCPOK, SR=SR,VTBI=VTBI,HH=HH,MM=MM, IR=IR})
({BC=BC,KP=KP,PL=PL,TL=TL, BD=BD,SS=SS, VL=VL,PP=PP,SOK=SOK,BCPOK=BCPOK, SR=SR,VTBI=VTBI,HH=HH,MM=MM, IR=IR})
Z
1
Z
Z
1`{BC=90,KP=no,PL=yes,TL=yes,BD=BDPlastipak,SS=10,VL=0,PP=10,SOK=yes,BCPOK=yes,SR=no,VTBI=10,HH=24,MM=0,IR=0}

Fig. 6: Info Page with Z

Z, assigns each variable to its *new* value, as given by the right-hand side of each equation in the *KeyPadLocked* operation schema. Taken together these two arcs express the intent of the equations in the operation schema *KeyPadLocked*. Users can go to any of the five states by firing the I-behaviour transitions which will update the markings on the corresponding places. If the value of KP is *yes*, i.e., if the keypad is locked, then a user cannot go to any further possible states. For that reason, we have a guard $[KP = no]$ on transitions *I_BatteryLevel*, *I_EventLog*, *I_Init*, *I_RateSet* and *I_ChangeSetUp*. These transitions would not be enabled if the keypad is locked. In a similar manner we model the rest of the pages[9] which are interconnected via fusion places and a user can go from one state to another by firing the I-behaviour transitions and the operations can be observed by firing the S-behaviour transitions.

8 Benefits of Formal Method Integration

There are several benefits accruing from using CPN and its tool and combining what was previously done via three different formalisms (PM/PIM/Z). First, CPNs have a simple graphical representation which is useful for illustrating the concepts, and the CPN Tool allows us to visualize the models and structure them in useful ways. The CPN model provides a better understanding of how the system will behave by means of interactive simulation which provides a way to walk through a CPN model, investigating different scenarios in detail and checking whether the model works as expected.

[9] See `https://github.com/sapnajaidka/NikiT34-CPN-Model`

It is possible to observe the effects of the individual steps directly in the graphical representation of the CPN model. The models in CPN can be used to specify different aspects (functional or control flow) of a system and can specify different types (concurrent, sequential or distributed) of a system, all in one model.

When we build a Coloured Petri Nets model, any non-determinism present can be exposed and can be corrected. The appealing graphical representation of Coloured Petri Nets allows us to consider all the navigational possibilities in a model. It gives a good indication of the complexity of the user-interface and its navigation by way of the number of places and transitions. Having aspects of Z modelled in CPN has benefits in the development process as a developer can have a better idea of the user interface and interactivity as well as the functionality of a modelled system.

9 Conclusion and Future Work

We have used CPNs to model the user-interface and interaction of a medical infusion pump. This shows the navigational properties of the system, but also allows us to include the underlying system functionality in the model as well. By means of simulation we can actually see what widgets are available and what happens when the user interacts with them. Also we can actually *see* how the behaviours change the underlying system functionality. By these means we can check to see if the model is working as expected. If it seems that it is not then we can look deeper and see what the flaws are in the model and what changes should be made to make the model work correct in all situations: this is the most important thing for safety-critical devices.

Now that we can build CPN models there are certain properties that can be *verified* with the CPN's state space analysis method. This method provides information about the dynamic properties of a system, for example, dead transitions, and dead markings. It also gives information about the fairness and liveness properties of a modelled system. Therefore it is possible to investigate the behaviour of the system in sophisticated and useful ways: this includes the safety requirements of the system. With the state space method, in conjunction with suitable queries, it is possible to verify that queries hold, so safety requirements (like detecting livelocks, total reachability, desired terminal states etc.) for safety-critical systems can be proved. In the future it would be interesting to see if the safety properties proposed by the FDA[10] can be proved using this combined model.

References

1. Arney, D., Jetley, R., Jones, P., Lee, I., Sokolsky, O.: Formal methods based development of a PCA infusion pump reference model: Generic infusion pump (GIP) project. In: High Confidence Medical Devices, Software, and Systems and Medical Device Plug-and-Play Interoperability, 2007. HCMDSS-MDPnP. Joint Workshop on. pp. 23–33. IEEE (2007)
2. Barbosa, P.E., Morais, M., Galdino, K., Andrade, M.F., Gomes, L., Moutinho, F., de Figueiredo, J.C., et al.: Towards medical device behavioural validation using petri nets. In: Computer-Based Medical Systems (CBMS), 2013 IEEE 26th International Symposium on. pp. 4–10. IEEE (2013)
3. Boudi, Z., Collart-Dutilleul, S., Khaddour, M., et al.: High level Petri Net modeling for railway safety critical scenarios. In: 10th FORMS-FORMAT symposium, Formal Methods for Automation and Safety in Railway and Automotive Systems. pp. p65–75 (2014)

[10] See `https://repository.upenn.edu/cgi/viewcontent.cgi?referer=&httpsredir=1&article=1938&context=cis_reports`

4. Bowen, J.A.: Formal specification of user interface design guidelines. Ph.D. thesis, Citeseer (2005)
5. Bowen, J., Reeves, S.: Using formal models to design user interfaces: a case study. In: Proceedings of the 21st British HCI Group Annual Conference on People and Computers: HCI... but not as we know it-Volume 1. pp. 159–166. British Computer Society (2007)
6. Bowen, J., Reeves, S.: Formal models for user interface design artefacts. Innovations in Systems and Software Engineering **4**(2), 125–141 (2008)
7. Bowen, J., Reeves, S.: Modelling user manuals of modal medical devices and learning from the experience. In: Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems. pp. 121–130. ACM (2012)
8. Bowen, J., Reeves, S.: Generating obligations, assertions and tests from UI models. Proceedings of the ACM on Human-Computer Interaction **1**(EICS), 5 (2017)
9. Campos, J.C., Harrison, M.: Modelling and analysing the interactive behaviour of an infusion pump. Electronic Communications of the EASST **45** (2011)
10. Derrick, J., Boiten, E.A.: Refinement in Z and Object-Z: Foundations and advanced applications (2014)
11. Dix, A.J., Runciman, C.: Abstract models of interactive systems. People and Computers: Designing the interface pp. 13–22 (1985)
12. Electronics, C.M.: Niki T34 syringe pump instruction manual. ref. 100-090SS Edition (2008)
13. Jacob, R.J.: Using formal specifications in the design of a human-computer interface. Communications of the ACM **26**(4), 259–264 (1983)
14. Jaidka, S., Reeves, S., Bowen, J.: Modelling safety-critical devices: Coloured Petri Nets and Z. In: Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems. pp. 51–56. ACM (2017)
15. Jensen, K.: Coloured Petri Nets: basic concepts, analysis methods and practical use, vol. 1. Springer Science & Business Media (2013)
16. Jensen, K., Kristensen, L.M., Wells, L.: Coloured Petri Nets and CPN tools for modelling and validation of concurrent systems. International Journal on Software Tools for Technology Transfer **9**(3-4), 213–254 (2007)
17. Kristensen, L.M., Christensen, S., Jensen, K.: The practitioner's guide to Coloured Petri Nets. International Journal on Software Tools for Technology Transfer (STTT) **2**(2), 98–132 (1998)
18. Kummer, O., Wienberg, F., Duvigneau, M., Köhler, M., Moldt, D., Rölke, H.: Renew–the reference net workshop. In: Tool Demonstrations, 21st International Conference on Application and Theory of Petri Nets, Computer Science Department, Aarhus University, Aarhus, Denmark. pp. 87–89 (2000)
19. Lakos, C.: From Coloured Petri Nets to Object Petri Nets. In: International Conference on Application and Theory of Petri Nets. pp. 278–297. Springer (1995)
20. Martinie, C., Navarre, D., Palanque, P., Fayollas, C.: A generic tool-supported framework for coupling task models and interactive applications. In: Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems. pp. 244–253. ACM (2015)
21. Masci, P., Ayoub, A., Curzon, P., Lee, I., Sokolsky, O., Thimbleby, H.: Model-based development of the generic PCA infusion pump user interface prototype in PVS. In: International Conference on Computer Safety, Reliability, and Security. pp. 228–240. Springer (2013)
22. Navarre, D., Palanque, P., Ladry, J.F., Barboni, E.: ICOs: A model-based user interface description technique dedicated to interactive systems addressing usability, reliability and scalability. ACM Transactions on Computer-Human Interaction (TOCHI) **16**(4), 18 (2009)
23. Rukšėnas, R., Masci, P., Harrison, M.D., Curzon, P.: Developing and verifying user interface requirements for infusion pumps: A refinement approach. Electronic Communications of the EASST **69** (2014)
24. Silva, J.L., Ribeiro, O.R., Fernandes, J.M., Campos, J.C., Harrison, M.D.: The APEX framework: prototyping of ubiquitous environments based on Petri Nets. In: International Conference on Human-Centred Software Engineering. pp. 6–21. Springer (2010)
25. Smith, G.: The Object-Z specification language, vol. 1. Springer Science & Business Media (2012)
26. Turner, J.D.: Supporting interactive system testing with interaction sequences. Ph.D. thesis, The University of Waikato (2019)

Fortune Nets for Fortunettes: Formal, Petri nets-based, Engineering of Feedforward for GUI Widgets

David Navarre1[0000-0002-2900-2056], Philippe Palanque1-2[0000-0002-5381-971X], Sven Coppers3[0000-0002-5734-8898], Kris Luyten3[0000-0002-4194-1101] and Davy Vanacken3[0000-0001-8436-5119]

[1] ICS-IRIT, University of Toulouse, 118 Route de Narbonne, F-31062, Toulouse, France
[2] Technical University Eindhoven, Department of Industrial Design, Eindhoven, Netherlands
[3] Hasselt University - tUL - Flanders Make, Expertise Centre for Digital Media, Diepenbeek Belgium
`{navarre, palanque}@irit.fr ; firstname.lastname@uhasselt.be`

Abstract. Feedback and feedforward are two fundamental mechanisms that supports users' activities while interacting with computing devices. While feedback can be easily solved by providing information to the users following the triggering of an action, feedforward is much more complex as it must provide information before an action is performed. Fortunettes is a generic mechanism providing a systematic way of designing feedforward addressing both action and presentation problems. Including a feedforward mechanism significantly increases the complexity of the interactive application hardening developers' tasks to detect and correct defects. This paper proposes the use of an existing formal notation for describing the behavior of interactive applications and how to exploit that formal model to extend the behavior to offer feedforward. We use a small login example to demonstrate the process and the results.

Keywords: Feedforward, formal methods, Petri nets, interactive systems engineering.

1 Introduction

Feedback and feedforward are two fundamental mechanisms supporting users' activities while interacting with computing devices. While feedback can be easily solved by providing information to the users following the triggering of an action, feedforward is much more complex as it must provide information before an action is performed. Automatic feedforward presents in a systematic way to the users what can be done without requiring any dedicated action (e.g. greying out an interactive object that is not available). Automatic feedforward is often available in well-designed interfaces. User-triggered feedforward provides localized, contextual information to the users related to the actions that they envision triggering (e.g. painting temporarily a selected object in yellow while hovering over the yellow button for painting objects). User-triggered feedforward is usually not available in user interface, as it requires computing the future state of the application (if a given action is performed) and presenting this future state on the UI.

In [25], the authors exploit Norman's activity theory [17] to explain the importance and the impact of providing users with feedforward in user interfaces, especially in the action selection phase. In poorly designed systems, that kind of user activity can be very cumbersome especially in the upper part of the model of the activity theory (also called semantic distance).

Fig. 1 presents a typical system offering limited feedforward. In that system (Microsoft Word) some of the commands for changing text graphical attributes do not propose feedforward (see **Fig. 1** b) while others do (see **Fig. 1** c)). **Fig. 1** a) presents a snapshot of MS Word software with the word Fortunettes selected and highlighted. In that version of MS Word, when some text is selected, a contextual pop-up menu appears next to the selected text. In **Fig. 1** a) the cursor has been moved far away from the selected text and thus no pop-up menu is visible. In **Fig. 1** b) the pop-up menu is displayed and the mouse cursor hovers over the Bold command to change the presentation of the text to Bold. However, in that case, no feedforward is presented so it is not possible to see how the text will be if the Bold command is performed. Surprisingly, **Fig. 1** c) highlights the fact that for altering the color of the selecting text, hovering over one of the colors displayed in the pop-up menu applies directly the hovered over color to the selected text, thus providing users with feedforward on the color attribute of the text.

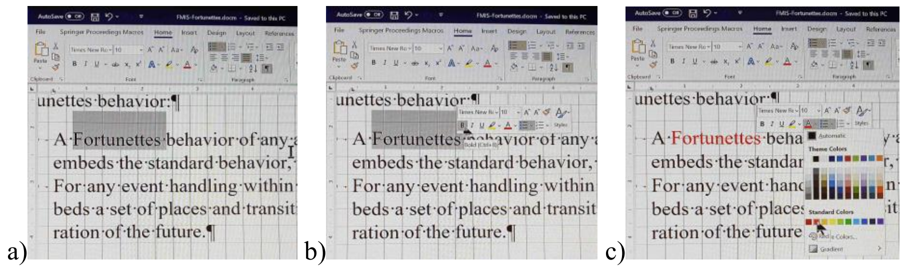

a)
b)
c)
unettes·behavior:¶
A·Fortunettes·behavior·of·any·
embeds·the·standard·behavior,·
For·any·event·handling·within·
beds·a·set·of·places·and·transit
ration·of·the·future.¶
Theme Colors
Standard Colors

Fig. 1. Inconsistent availability of feedforward in Microsoft Word (Office 2016)

One of the questions that arises immediately is: why such a sophisticated tool as MS Word is not offering feedforward mechanisms for all the functions or at least to all the similar functions (e.g. changing attributes of selected text).

While, as highlighted in [10] and [25], the design of feedforward is an issue. We would argue that its specification and its implementation are the key problem to solve when it is considered as a potential function to add to the system. In that case, we would argue that feedforward is a **usability function** using the pending concept of **security function** [26] or **safety function** [13]. While a safety function can be defined as a function added to a system to prevent undesired safety problems, we would define a usability function as a function added to an interactive system to prevent undesired usability problems. Within this context, feedforward can be considered as a function similar to "undo" and thus requires complex implementation due to its crosscutting nature [13].

This paper argues for the use of a formal approach for the specification and the implementation of feedforward in a systematic way. We present how the expressive power of high-level Petri nets such as ICOs [16] can describe feedforward and how the resulting models are amenable to verification (to identify and check properties on the system offering feedforward). In a nutshell we propose to produce a Petri net model (called *Fortune Net*) in addition to the model describing the behavior of the application. We also argue that a formal model of the initial application can be extended in a systematic way to include feedforward functionality, thus reducing development cost of such a usability function.

This paper is an extension of the work done in [7] to offer feedforward mechanisms in a more general context. Section 2 presents the foundations, interaction and one design for the Fortunettes concept for feedforward usability function. Section 3 presents the illustrative example of a simple widget-based interactive application that is used throughout the paper. Section 4 presents the Petri nets based modeling approach for modeling interactive applications and its application to the modelling of Fortunettes usability function. Section 5 focusses on the formal analysis of the application model and of the Fortune Nets ones. Section 6 concludes the paper and highlight paths for future work.

2 Fortunettes: Design, Foundations and Use

The origin of Fortunettes [7] is the need of providing feedforward about the future state of an application. When including a feedforward usability function in the GUI, the feedforward information does not need to be presented permanently (to avoid visual overload and cluttering of the UI) but instead we propose this specific information display to be triggered by the user on demand (when needed). In our approach, exploring the future may be seen as a four steps process:

- **Look at the present,** when the user explores visually the user interface elements;
- **Peek into the future**, when the user is considering performing an action;
- **Go to the future**, when the user confirms and actually executes the considered action;
- **Return to the present**, when the user is no longer considering the execution of that action.

The choice has been made of providing such feedforward at widget-level as it makes it easier to reuse for any widget-based application. **Fig. 2** shows an example of this kind of widget-level feedforward: when the user hovers over the `Login` button (that is currently enabled), the button `Logout` and the text box (that are currently disabled), show their future state in terms of availability (the button `Logout` and the textbox will become enabled if the user clicks the button `Login`, while the button `Login` will become disabled). With this information, the user knows that to enable the `Logout` button, the `Login` button me be pressed first.

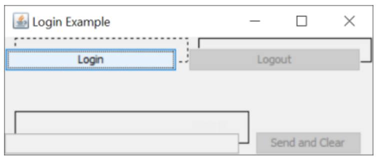


Fig. 2. Illustration of the Fortunettes concept using the case study.

The main idea of Fortunettes is to provide the user with an answer to "What will happen if I do that?", by presenting what the result of the user action will be, before the action is actually performed. It thus requires the widgets to be able to present their future state in addition to their current state (enabled or disabled).

As presented in [7], the user interface of the application presented in **Fig. 2** is the following:

- The application is composed of four widgets (the three buttons and a textbox),
- The current state of the widgets is displayed on the forefront, the login button is enabled, "Logout" and "Send and Clear" ones are disabled, and the textbox is disabled too.
- In order to present the feedforward information, users have to hover over the widget of interest. In **Fig. 2**, the "login" button is hovered and the background display of each widget presents the feedforward information showing the state of the application if the user clicks on the login button. Current feedforward display tells the user that "login" button will be disabled, the textbox will become enabled, "logout" will become enabled and "Send and Clear" will remain disabled. Indeed, as the status of "Send and Clear" will remain the same, no additional feedforward display is presented. We follow here the parsimony principle of user interface designs.

The design choice presented here is one example of the many possible designs of Fortunettes: every widget is decorated with borders to express its future availability (full lines for enabled, dashed lines for disabled) and/or its future values.

This design will not be further discussed as the focus of this paper is on formal description and engineering support. These two aspects are particularly important as the introduction of Fortunettes increases the complexity of the development of an application, and, by consequence its reliability.

3 Illustrative example

We illustrate the use of the Fortunettes approach with a simple application (as illustrated by **Fig. 3**) that behaves as follows: when the user is logged in, a message can be written in the textbox or the user can log out. To ensure that the message only contains letters, the edited text is filtered, removing any other characters (numbers, special characters…). If the textbox is not empty, the message can be sent. Sending the message or logging out clear the textbox.

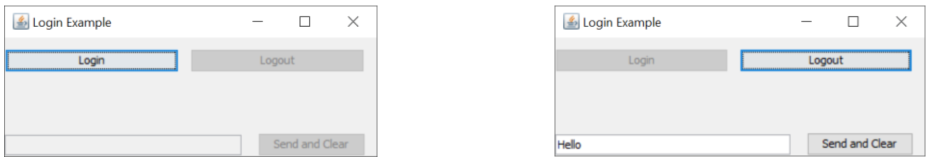


Fig. 3. Screen shots of the illustrative application. On the left, the user is logged out, on the right, the user is logged in and a message is being edited and ready to be sent.

4 Modelling of Fortunettes behavior

To support the engineering of interactive applications offering a feedforward usability function based on Fortunettes, we propose an approach based on a formal description technique called Interactive Cooperative Objects (ICO). We firstly present in this Section the formal description technique, then we present how it is possible to derive the feedforward behavior of the application from the existing model of the application behavior.

4.1 ICO formal description technique

The ICO formalism is a formal description technique dedicated to the modeling and the implementation of event-driven interfaces [16], using a decomposition of communicating objects to model the system, where both behavior of objects and communication protocol between objects are described by the Petri net dialect called Cooperative Objects (CO) [4]. In the ICO formalism, an object is an entity featuring four components: a cooperative object which describes the behavior of the object, a presentation part (i.e. the graphical interface), and two functions (the activation function and the rendering function) which connects the cooperative object and the presentation part.

An ICO specification fully describes the potential interactions that users may have with the application. The specification encompasses both the "input" aspects of the interaction (i.e. how user actions impact the inner state of the application, and which actions are enabled at any given time) and its "output" aspects (i.e. when and how the application displays state information that is relevant to the user).

This formal description technique has already been applied in the field of Air Traffic Control interactive applications [16], space command and control ground systems [20], or interactive military [3] or civil cockpits [2].

The ICO notation is fully supported by a CASE tool called PetShop [5, 21]. All the models presented in the two next Sections (4 and 5) have been edited, simulated and analyzed using PetShop tool.

4.2 Principle of Fortunettes feedforward modelling using ICO

As stated in Section 2, engineering an application with feedforward capabilities requires to handle extra interaction events (at least three, depending on the widget type). These events allow the user to **peek into the future**, to **go to the future** or to **return to the present**, without affecting the standard behavior of the application, as the objective is to enhance the application (with feedforward) and not to change it. This design choice impacts the modelling of feedforward behavior:

- The feedforward behavior of any application is modelled as an independent object that embeds the standard behavior (as a copy), making it fully compatible with the original application behavior. This Petri net model is called the **Fortune Net** as it allows users to look into the future of the application.
- For any event handling within the standard behavior, the feedforward behavior embeds a pattern described in Petri nets (a set of places and connected transitions) that models the exploration of the future states. The important aspect in this modelling principle is that we exploit the behavior of the application to forecast the future states of the application if the user decides to use feedforward function.

To illustrate these two points, we use an excerpt of the complete behavior presented in the next Section (4.3) that only concerns the `login` action on the user interface (as shown by **Fig. 4**).

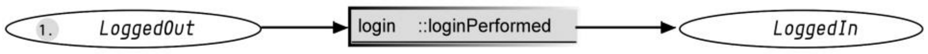


Fig. 4. Excerpt from the Petri net model of the standard behavior of the application: event handling of the login action. In the transition, the text on the left describes the name of the transition while the text on the right describes the name of the event associated the transition.

In Fig. 4, the login transition is the event handler for an event called `loginPerformed` that represents the use of the button Login. When fired, this transition moves the token from place `LoggedOut` (1.) to place `LoggedIn`, setting the state of the application to the new state following the execution of the login (code not represented here).

When introducing the Fortunettes view on this action, the three base actions defined in Section 2 (peek into the future, go to the future and return to the present) are represented as three extra event handlers, as shown on Fig. 5, where event handlers {`FloginPerformed`, `UFloginPerformed`, `InFloginPerformed`} are generated from the event handler `loginPerformed`. In this paper, the name of the generated event handlers for handling Fortunettes mode are built with the name of the corresponding event handler, prefixed by `F` (that represents entering in Fortunettes mode, e.g. peek into the future), by `UF` (that represents exiting the Fortunettes mode, e.g. return to the present) and `InF` (that represents exiting the Fortunettes mode and go to the future).

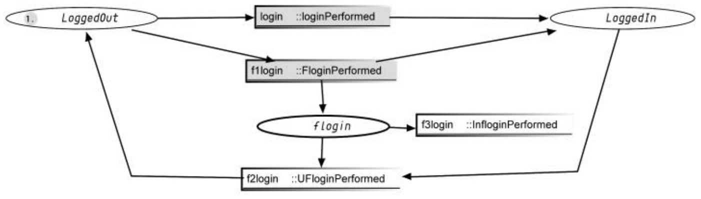

1.
LoggedOut
login ::loginPerformed
LoggedIn
f1login ::FloginPerformed
flogin
f3login ::InfloginPerformed
f2login ::UFloginPerformed

Fig. 5. Extracted from the feedforward behavior of the application: event handling of the login action and peek into its future.

On Fig. 5, transition `f1login` (event handler for `FloginPerformed`) represents the action of peeking into the future of the action `login`. Basically, it behaves in the same way as the original action (put a token in place `LoggedIn`) while the standard behavior is still in state `LoggedOut`. It additionally puts a token in place `flogin` that represents the entering in feedforward mode (a dedicated rendering may occur).

There are then two possibilities:

- The user decides to really perform the `login` action (using the `login button`), producing two events: `loginPerformed` handled by the standard behavior (making it going to the state `LoggedIn`) and `InFloginPerformed` handled by the feedforward behavior (discarding the token in place `flogin`, while the token in place `LoggedIn` does not move, placing it in the same state as the standard behavior).
- The user decides to not perform the login action producing an event `UFloginPerformed`. The standard behavior remains in the same state while in the feedforward behavior, the tokens from places `LoggedIn` and `flogin` are removed and a token goes back to the place `LoggedOut`, making it return to its previous state (leaving the feedforward mode).

This pattern is particularly efficient when describing a feedforward behavior for events that do not handle values or when the widgets are simple such as button. For more complex events, or when the underlying widgets are more complicated, this pattern has to be modified/extended:

- When values are handled by the action of the widget, it is not always possible to peek into the future of these values. One possible improvement is to proceed in two steps. When entering the feedforward mode, an envisioned value must be produced (decided at design time for instance) and when the user really performs the action, a substitution must be done between the envisioned value and the real value. In the feedforward behavior, this can be done by moving tokens (if it was the case in the login example, the first token put in place `LoggedIn` by transition `f1login` would have a design time envisioned value, and when `f3login` would be fired, this token would have been removed and replaced by one holding the correct value).
- When the widget is more complex (in our case, the complexity is related to the event production), extra event handlers may be introduced. For instance, when using a classical textbox, one may be interested by the end of the text edition (validation) and not by the whole process of typing in the text. In this case, in the standard behavior of the application, the only handled event would be the last one (for instance, the event `actionPerformed` of the `JTextField` in Java Swing). On the feedforward behavior side, any text change may be relevant to allow the rendering of text filtering.

Fortunettes requires enhancing widgets with extra means to allow rendering feedforward states and to trigger dedicated events. In our implementation using Java Swing widgets, we embed them within a specialized decorator, but there are many other implementation options at widget level or at application level.

4.3 Application of the modeling principle to the illustrative example

This Section presents the ICO models for both the standard application and its Fortunettes enhancement. For each model, we present the behavioral part and the two user interface description functions: the activation part and the rendering part.

Standard behavior.

Fig. 6 presents the entire behavior of the illustrative example. It may be divided into two parts: the upper part is dedicated to login actions and the lower part is dedicated to the message handling.

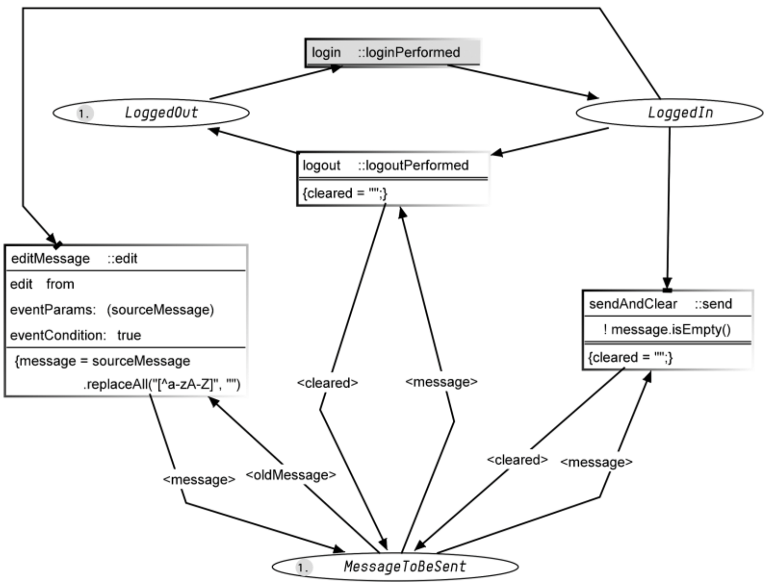


Fig. 6. Behavior of the Login example using the ICO formal description technique.

The upper part of **Fig. 6** models what has been explained in the beginning of the Section (see Fig. 4) to introduce Fortunettes and the modelling approach, including the complete behavior of the application i.e. its functional code (inside the transitions). Another difference is the way back from place `LoggedIn` to place `LoggedOut` when logging out that clears the edited message (modification of the value of the token held by place `MessageToBeSent).`

The lower part of **Fig. 6** is dedicated to the message editing and to send it. Sending it (transition sendAndClear) can only occur if the message is not empty (precondition `!message.isEmpty()`). When it occurs, the token held by place `MessageToBeSent` is destroyed and a new token (with an empty string) is set to that place. The message editing is represented by the transition `editMessage` that receives an event called `edit`, and this event holds a string value called `sourceMessage`. This `sourceMessage` is then filtered resulting in a string `message` that only contains characters that belongs to `[a-z]` and `[A-Z]` (For instance "`a1b2c3`" will be transformed into "`abc`") by the execution of the function replaceAll.

Table 1 represents the activation function of the application. It relates the event production from the application and event handlers described using ICO. When the event occurs, the corresponding transition is fired. If the transition is not available, the corresponding event source must be disabled. This part of the functioning is assumed by the activation rendering method (last column of **Table 1**) that is provided by the application: for instance, `setLoginEnabled` changes the enabling of the button `Login`.

Table 1. Activation function for the ICO model of the Login example.

User Event	Event handler	Activation Rendering
Edit	editMessage	setEditEnabled
Login	login	setLoginEnabled
Logout	logout	setLogoutEnabled
Send	sendAndClear	setSendEnabled

Table 2 represents the rendering function of the application. It relates any state change within the application behavior to rendering methods call. For instance, when a token enters place `MessageToBeSent`, the string of this message is set in the text box widget by calling the method `showMessage`.

Table 2. Rendering function for the ICO model of the Login example.

ObCS node name	ObCS event	Rendering method
MessageToBeSent	marking_reset	showMessage
MessageToBeSent	token_enter	showInitialMessage

Feedforward behavior.

Fig. 7 illustrates how feedforward information can be displayed using Fortunettes. **Fig. 8**, **Table 3** and **Table 4** fully describe the feedforward part of the application. The behavior presented by **Fig. 8** is structured similarly to the standard behavior, the upper part being dedicated to the login actions and the lower part, to the message editing.

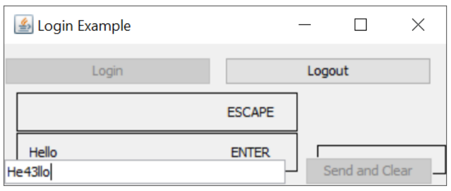


Fig. 7. Illustration of the text filtering while typing in feedforward mode

This Fortune Net behaves according to the pattern explained in the previous Section with the particularity of the filtering of the text while it is being typed in and not only at the end of the interaction with the text box (transition `f4editMessage` in the lower part of **Fig. 8**). This allows to present to the user what will happen to the edited value if it is validated (e.g. press ENTER), as illustrated by **Fig. 7**.

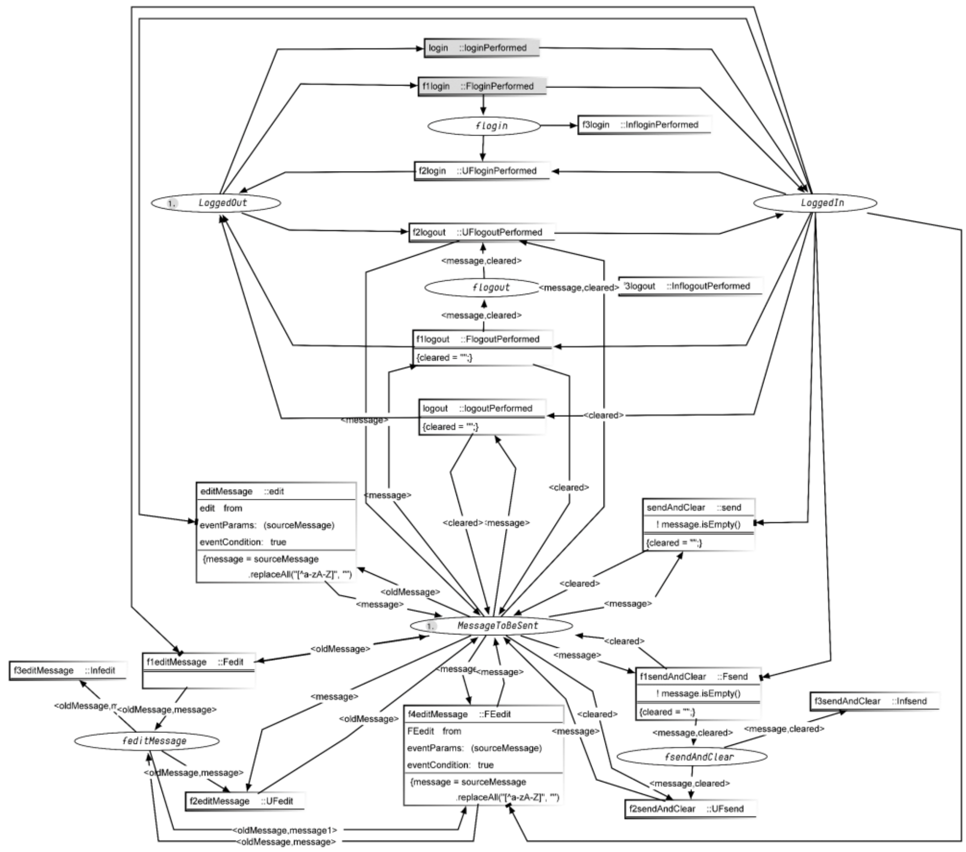


Fig. 8. The Fortune Net describing the feedforward behavior of the Login example using the ICO formal description technique.

Table 3 presents the activation of the feedforward behavior of the application. The interesting part of this function is that the activation rendering is not related to the immediate availability of the events, but to their availability in the future. Therefore, it does not directly impact the application widgets but only calls functions that have been added to render their Fortunettes appearance. For instance, on **Fig. 7**, if the edited text is validated (e.g. pressing ENTER), the button “Send and Clear” will become available (represented by the rectangle around it, in the background).

Table 3. Activation function for the ICO model of the feedforward behavior of the example.

User Event	**Event handler**	**Activation Rendering**
Edit	editMessage	setFortunettesEditEnabled
Login	login	setFortunettesLoginEnabled
Logout	logout	setFortunettesLogoutEnabled
Send	sendAndClear	setFortunettesSendEnabled

Table 4 presents the rendering function of the feedforward behavior of the application. This function first aims at making the application entering in feedforward mode (a token enters any of the places prefixed `f`) or at exiting the feedforward mode (a token exits any of the paces prefixed by `f`). This function ensures too that when a new message is under editing, it is rendered on the feedforward part of the interface (each time a token enters the place `MessageToBeSent`, `showFortunettesMessage` is called modifying what is rendered in the `ENTER` rectangle of the text box as illustrated on Fig. 7)

Table 4. Rendering function for the ICO model of the feedforward behavior of the example.

ObCS node name	**ObCS event**	**Rendering method**
MessageToBeSent	marking_reset	showFortunettesMessage
MessageToBeSent	token_enter	showFortunettesInitialMessage
fEditMessage	token_enter	startRenderFortunettes
fEditMessage	token_exit	stopRenderFortunettes
fLogin	token_enter	startRenderFortunettes
fLogin	token_exit	stopRenderFortunettes
fLogout	token_enter	startRenderFortunettes
fLogout	token_exit	stopRenderFortunettes
fSendAndClear	token_enter	startRenderFortunettes
fSendAndClear	token_exit	stopRenderFortunettes

This interesting joint behavior between the standard behavior of the application and its Fortunettes ones is highlighted on **Fig. 7**. Indeed, when the user types some text in, it is rendered directly in the text box while the Fortunettes rendering displays the text, as it will appear if the validation key is pressed. In the case of the login application, we see that all the non-textual characters will be removed and the current text “He43llo” will appear as “Hello” in the future.

5 Formal Analysis on the illustrative example

This Section is dedicated to the formal analysis of the models presented above. The fact that we produce two different models for the same application (the standard application model and the Fortune Net) has multiple implications. First, the standard application models must exhibit some properties and it is important to check that they are true. Second, the Fortune Net also needs to exhibit some properties (e.g. each time the user triggers the “peek into the future” there must be two actions available: one to go into that peeked future and one to come back to the current present. Third, the Fortune Nets must implement a “similar” behavior as the standard application and thus we must demonstrate their compatibility. For instance, it is important to demonstrate that all the

actions available in the models of the standard application are available in the Fortune Net. This is only an example of the generic properties that have to be checked when a feedforward usability function is added to an application.
With ICOs, as detailed in [24] and [19], there are two different techniques:

- The analysis of the underlying Petri net using results from Petri nets theory. This analysis can be performed using methods and algorithms from the Petri nets community such as the ones presented in [15].
- The analysis of the high-level Petri net (ICO) but this requires manual demonstrations as some of the properties are undecidable [9].

Due to space constraints, we only present here properties that are based on the underlying Petri net model. Some interesting results demonstrate that the high-level nature of the Petri nets with objects only reduce the availability of transitions (for instance when they feature pre-conditions) and thus in order for the high-level Petri net to be live, the underlying Petri net must be live [4].

5.1 Formal analysis of the model of the standard behavior (Fig. 6)

Table 5 presents the list of traps and siphons of the model in **Fig. 6**[1]. In a Petri net a siphon is a set of places that never gain tokens whatever transition is fired while a trap is a set of places that never lose tokens [8]. The fact that all the places in the model are both traps and siphons demonstrate that the number of tokens in the model will remain the same as the one in the initial state i.e. two tokens (see **Fig. 6**).

Table 5. Siphons and Traps from the standard behavior of the application.

Siphons	Traps
MessageToBeSent	MessageToBeSent
LoggedIn, LoggedOut	LoggedIn, LoggedOut

Table 6 analysis is based on the calculation of transition invariants and place invariants. As can be seen all the places in the model belong to a place invariant which means that the total number of tokens in the places of the models will remain the same. One interesting piece of information is that place MessageToBeSent is a single place in a P-invariant. This means that whatever transition is fired the number of tokens in that place will always be the same as the one of the initial marking. In the current example, this means that the place MessageToBeSent will always be marked by a single token.

Table 6. Transitions and Place Invariants from the standard behavior of the application.

T-Invariants	P-Invariants
1 sendAndClear	1 LoggedIn, 1 LoggedOut
1 editMessage	1 MessageToBeSent
1 login, 1 logout	

[1] The computing of the results in those tables was done using Petshop tool and are not presented due to space constraints. How to make such computing is presented in [8].

In terms of behavior, transitions login and transition logout belong to the same t-invariant which means that, if they can be made available from the initial state, there always exists a sequence of transitions in the Petri net to make them available. Their connection with the P-invariant {1 LoggedIn, 1 LoggedOut} (with a bounded value of one token) demonstrates that always one of the two transition will be available and they will never be available at the same time.

5.2 Formal analysis of the Fortune Net (Fig. 8)

We will not detail the analysis of the Fortune Net, but it is important to check that the properties true in the application model are still holding in the Fortune Net.

If we take as example the property of the mutual exclusion of login and logout transitions, we can easily see in **Table 7** and **Table 8** that the places and the transitions belong are also listed in siphons, traps, P-invariants and T-invariants.

Table 7. Siphons and Traps from the feedforward behavior of the application.

Siphons	Traps
MessageToBeSent	MessageToBeSent
LoggedIn, LoggedOut	LoggedIn, LoggedOut

Of course, the Fortune Net is more complex and should also exhibit specific properties related to its own semantics. A very simple but important one is that whenever the user triggers a transition to peek into the future (name starting with f1) immediately after a transition to come back to present (name starting with f2) and a transition to go into the future (name starting with f3) will be available. The analysis results in **Table 8** demonstrate that a Fortune Net always verifies this fundamental property (any of such transitions is always in a T-Invariant with each other).

Table 8. Transitions and Place Invariants from the feedforward behavior of the application.

T-Invariants	P-Invariants
1 f4editMessage	1 LoggedIn, 1 LoggedOut
1 f1logout, 1 f3logout, 1 login	1 MessageToBeSent
1 f1login, 1 f2login	
1 editMessage	
1 f1editMessage, 1 f2editMessage	
1 f1sendAndClear, 1 f3sendAndClear	
1 f1sendAndClear, 1 f2sendAndClear	
1 f1logout, 1 f2logout	
1 login, 1 logout	
1 f1login, 1 f3login, 1 logout	
1 f1login, 1 f1logout, 1 f3login, 1 f3logout	
1 sendAndClear	

1 f1editMessage, 1 f3editMessage	
1 f1login, 1 f1logout, 1 f2login, 1 f3logout, 1 login	
1 f1login, 1 f2login, 1 login, 1 logout	

6 Related work

As highlighted in [22] many formal approaches to support the design, specification and verification of interactive systems have been proposed. That book chapter highlights four criteria to compare those approaches: 1) Modeling coverage (how much of the interactive systems can the notation describe); 2) Properties (and their type) supported; 3) Application of the methods in which domain; 4) Scalability (is the notation able to deal with large scale interactive systems).

With respect to the modelling need of Fortunettes, the expressive power of the notation to be used heavily depends on the interactive application itself and does not require specific modelling power. With that respect, if the interactive application does not feature concurrent behavior, dynamic instantiation of objects and does not exhibit quantitative time behavior, automata would be adequate for describing Fortunettes behavior as demonstrated in [7]. If more complex behaviors need to be represented, more expression power will be required. The table 1 from the book chapter [22] would be then of great help to select the modeling notation.

As Fortunettes feedforward concept is meant to be applied in a systematic way to all the interactions in an interactive system, Fortune Nets need to cover all the aspects of the interactive (from the low-level interaction technique to the functional core according to the MIODMIT architecture [14]. We have only presented here Fortunettes at the application level, but all the layers of the architectures should be taken into account.

7 Conclusion and perspectives

While research in the field of HCI focuses on adding more functionalities to the user interface, the interaction techniques and the interactive applications to improve usability and user experience, very little work is spent on transferring these improved interactions to the developers of interactive systems. For instance, papers proposing bubble cursor for improving target acquisition [11] or marking menus [12] to improve command selection do not present means for engineering these interaction techniques in a reliable and systematic way.

This paper has proposed an engineering method based on formal methods to support the systematic integration of Fortunettes concepts to provide interactive application with feedforward mechanisms. While the graphical and interaction design of Fortunettes might be improved and could be subject of future research, we have demonstrated that the use of a Petri nets-based approach limits the complexity of adding Fortunettes behavior to an existing application. We have also demonstrated that a formal approach can provide benefits in ensuring that the application with the additional feedforward behavior remains behaviorally compatible with the initial application.

The work presented in the present paper leads to extensions that should be addressed in future work. First, the current design of Fortunettes only deals with WIMP interaction techniques based on a set of identified widgets. While this can be seen as a strong limitation for current user interfaces targeting at better user experience, it is important to note that many applications are still widget-based. In some critical domains it is even not possible to embed other types of interfaces as required by the ARINC 661 specification standard [1] for user interfaces of cockpits of large civil aircrafts. We have previously worked on the formal description of User Application, user interface widgets and servers using Petri net based description [2] and that early work can directly benefit from the work presented in the paper. This means that adding the feedforward usability function to those user applications will result in very limited work (as the Fortune Net is built upon the original behavior and is described with the same language) and would come with assurance means to guarantee their correct behavior.

Second, the current behavior of Fortunettes is to offer the possibility to the user to look only one step into the future. The model-based behavior presented in the paper could be exploited further to look into several step or even to look at the eventual end of the execution, as introduced in [19]. For instance it would be possible to identify a widget (via formal analysis) that would become unavailable forever in five steps from the current state of the application .While graphical design and interaction will be clearly a difficult challenge, the engineering of such applications could be reachable via the analysis of the formal models.

References

1. ARINC 661. Cockpit Display System Interfaces to User Systems. ARINC Specification 661-5. AEEC, 2013
2. Barboni E., Conversy S., Navarre D. & Palanque P. Model-Based Engineering of Widgets, User Applications and Servers Compliant with ARINC 661 Specification. 13th conf. on Design Spec. and Verif. of Interactive Systems (DSVIS 2006), LNCS Springer Verlag. p25-38
3. Bastide R., Navarre D., Palanque P., Schyn A. & Dragicevic P. A Model-Based Approach for Real-Time Embedded Multimodal Systems in Military Aircrafts. Sixth International Conference on Multimodal Interfaces (ICMI'04) October 14-15, 2004, USA, ACM Press.
4. Bastide R., Sibertin-Blanc C., Palanque P. Cooperative objects: A concurrent, petri-net based, object-oriented language. IEEE Systems Man and Cybernetics Conference-SMC 1993, 286-291
5. Bastide, R., Navarre, D., Palanque, P.: A Model-based Tool for Interactive Prototyping of Highly Interactive Applications. CHI '02 Extended Abstracts on Human Factors in Computing Systems. pp. 516–517. ACM, , USA (2002).
6. Canfora G. and Luigi Cerulo. 2005. How Crosscutting Concerns Evolve in JHotDraw. In Proceedings of the 13th IEEE International Workshop on Software Technology and Engineering Practice (STEP '05). IEEE Computer Society, Washington, DC, USA, 65-73.
7. Coppers, S., Luyten, K., Vanacken, D., Navarre, D., Palanque, P., Gris, C. Fortunettes: Feedforward about the Future State of GUI Widgets. Proceedings of the ACM on Human-Computer Interaction vol:3. ACM SIGCHI.
8. David R., Alla H. Petri nets and grafcet - tools for modelling discrete event systems. Prentice Hall 1992, ISBN 978-0-13-327537-7, pp. I-XII, 1-339

9. Dietze R., Kudlek M., Kummer O. Decidability Problems of a Basic Class of Object Nets. Fundam. Inform. 79(3-4): 295-302 (2007)
10. Djajadiningrat T., Kees Overbeeke, and Stephan Wensveen. 2002. But how, Donald, tell us how?: on the creation of meaning in interaction design through feedforward and inherent feedback. Conference on Designing interactive systems: processes, practices, methods, and techniques (DIS '02). ACM, New York, NY, USA, 285-291.
11. Grossman T. and Balakrishnan R. 2005. The bubble cursor: enhancing target acquisition by dynamic resizing of the cursor's activation area. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '05). ACM, DL, 281-290.
12. Kurtenbach G. and Buxton W. 1994. User learning and performance with marking menus. Conference on Human Factors in Computing Systems (CHI '94). ACM DL, 258-264.
13. Lee S. & Yamada Y. (2010) Strategy on Safety Function Implementation: Case Study Involving Risk Assessment and Functional Safety Analysis for a Power Assist System, Advanced Robotics, 24:13, 1791-1811
14. Martin Cronel, Bruno Dumas, Philippe A. Palanque, Alexandre Canny. 2018. MIODMIT: A Generic Architecture for Dynamic Multimodal Interactive Systems. In Proc. of IFIP TC13.2 Conference on Human Centered Software Engineering, HCSE 2018, 109--129.
15. Murata T. Petri nets: Properties, analysis and applications. Proceedings of the IEEE (Volume: 77 , Issue: 4 , Apr 1989)
16. Navarre D., Palanque P., Ladry J-F. & Barboni E. ICOs: A model-based user interface description technique dedicated to interactive systems addressing usability, reliability and scalability. ACM Trans. Comput.-Hum. Interact., 16(4), 18:1–18:56. 2009.
17. Norman, D. A. The Psychology Of Everyday Things. Basic Books, New York, USA, June 1988
18. Palanque P., Bastide R., Dourte L.Contextual Help for Free with Formal Dialogue Design. In Proceedings of HCI International (2) 1993: 615-620
19. Palanque P., Bastide R., Sengès V. Validating interactive system design through the verification of formal task and system models. IFIP WG 2.7, working conference Engineering HCI, 1995, Springer,189-212
20. Palanque P., Bernhaupt R., Navarre D., Ould M. & Winckler M. Supporting Usability Evaluation of Multimodal Man-Machine Interfaces for Space Ground Segment Applications Using Petri net Based Formal Specification. Ninth Int. Conference on Space Operations, Italy, June 18-22, 2006.
21. Palanque P., Ladry J-F, Navarre D. & Barboni E. High-Fidelity Prototyping of Interactive Systems can be Formal too 13th Int. Conf. on Human-Computer Interaction (HCI International 2009) LNCS, Springer.
22. Raquel Oliveira Prates, Philippe A. Palanque, Benjamin Weyers, Judy Bowen, Alan J. Dix. State of the Art on Formal Methods for Interactive Systems. Handbook of Formal Methods in Human-Computer Interaction 2017: 3-55
23. Sadasivan S., Joel S. Greenstein, Anand K. Gramopadhye, and Andrew T. Duchowski. 2005. Use of eye movements as feedforward training for a synthetic aircraft inspection task. Conference on Human Factors in Computing Systems (CHI '05). ACM, 141-149.
24. Silva J-L, Fayollas C., Hamon A., Palanque P., Martinie C., Barboni E. Analysis of WIMP and Post WIMP Interactive Systems based on Formal Specification. ECEASST 69 (2013)
25. Vermeulen J., Kris Luyten, Elise van den Hoven, and Karin Coninx. 2013. Crossing the bridge over Norman's Gulf of Execution: revealing feedforward's true identity. SIGCHI Conference on Human Factors in Computing Systems (CHI '13). ACM, USA, 1931-1940
26. Yoon C., Taejune Park, Seungsoo Lee, Heedo Kang, Seungwon Shin, and Zonghua Zhang. 2015. Enabling security functions with SDN. Comput. Netw. 85, C (July 2015), 19-35.

Model-Based Testing of Post-WIMP Interactions Using Object Oriented Petri-nets

Alexandre Canny[1], David Navarre[1], José Creissac Campos[2] and Philippe Palanque[1]

[1] ICS-IRIT, Université Paul Sabatier – Toulouse III, Toulouse, France
[2] HASLab/INESC TEC & Department of Informatics/University of Minho, Portugal
{alexandre.canny,navarre,palanque}@irit.fr, jose.campos@di.uminho.pt

Abstract. Model-Based Testing (MBT) relies on models of a System Under Test (SUT) to derive test cases for said system. While Finite State Machine (FSM), workflow, etc. are widely used to derive test cases for WIMP applications (i.e. applications depending on 2D widgets such as menus and icons), these notations lack the expressive power to describe the interaction techniques and behaviors found in post-WIMP applications. In this paper, we aim at demonstrating that thanks to ICO, a formal notation for describing interactive systems, it is possible to generate test cases that go beyond the state of the art by addressing the MBT of advanced interaction techniques in post-WIMP applications.

Keywords: Post-WIMP Interactive Systems, Software Testing, Model-Based Testing.

1 Introduction

Model-Based Testing (MBT) of software (called SUT: System Under Test) relies on explicit behavior models of a system to derive test cases [28]. The complexity of deriving comprehensive test cases increases with the inner complexity of the SUT that requires description techniques with an important expressive power. The modelling of post-WIMP (Windows, Icons, Menus and Pointers) interactive applications (i.e. applications with an interface not dependent on classical 2D widgets such as menus and icons [29]) proves to be a challenging activity as pointed out by [14]. For instance, when using a touch screen, each finger down/up is a virtual input device being added or removed from the systems at runtime and behaves in parallel with the other fingers or input devices. A modelling technique able to describe such interactive systems must support the description of dynamicity.

Beyond the problem of describing the SUT behavior, testing Graphical-User Interface, whether it is WIMP or post-WIMP, is known to be a complex activity [9], especially because of the unpredictability of the human behavior as well as the virtually infinite number of possible interaction sequences. To face such difficulty, model-based testing techniques have been developed to try to generate relevant test sequences without relying on manual scripting or capture and replay of tester's interactions.

The massive adoption of touch screens means advanced touch interactions (e.g. swipe, pinch-to-zoom, etc.) gained in popularity, while most of the existing MBT techniques for interactive applications are designed to deal with events performed on the standard GUI widgets (e.g. button, combo box, etc.) [1][9][15][27]. Lelli et al. [15] identified the need for new MBT techniques for post-WIMP applications by highlighting the need for supporting ad-hoc widgets (i.e. non-standard widgets developed specifically for the application) and advanced interaction techniques.

In this paper, we propose to build upon the work of Hamon et al. [14], which used the ICO [20] formal modelling technique to describe post-WIMP interactive systems, as a support to the generation of test cases for interaction techniques of post-WIMP applications and to demonstrate that testing can be conducted following the standard process for Model-Based Testing proposed in [28]. As interaction techniques have to cope with the high dynamicity of Input/Ouput, as well as temporal aspects, they prove to be one of the most difficult components of interactive systems to be described. Thus, they are the prime focus of this paper, even though we will highlight that our proposed approach applies to other components of the interactive systems' architecture as well.

This paper is structured as follows: Section 2 presents related work on the MBT of interactive applications; Section 3 introduces the interaction technique on which we propose to apply the approach and its modelling in ICO; Section 4 discusses the generation of the test cases from the ICO specification and Section 5 provides some comments on test execution; Section 6 discusses the generalizability of the proposed approach to other components of the SUT; Section 7 concludes the paper by discussing future work.

2 Related Work

The classical approaches to interactive applications testing consider that the user's interaction takes place at the GUI widget level (e.g. buttons, icons, label, etc.). While it is the case in the WIMP paradigm, this assertion cannot be used in the post-WIMP paradigm where "at least one interaction technique is not dependent on classical 2D widgets such as menus and icons" [29]. Consider a gesture-based (post-WIMP) drawing tool. One may want to define (and test) whether moving two fingers on the drawing area means zooming (pinch-to-zoom), rotating or drawing. As this may be determined by how the user effectively moves his/her fingers (speed, angle, pressure level, delay between finger down events, etc.), it goes beyond available standard testing techniques for widget level interactions.

In this section, we first introduce the process of MBT and discuss the existing Model-Based Testing techniques for WIMP applications. We then discuss the testing of post-WIMP applications in order to highlight challenges to overcome.

2.1 The Process of Model-Based Testing

In their Taxonomy of Model-Based-Testing Approaches, Utting et al. [28] present the model-based testing process illustrated by **Fig. 1**. In this process, a model of the SUT is built from informal requirements or existing specification documents (**Fig. 1**.(1)) and test selection criteria (**Fig. 1**.(2)) are chosen to guide the automatic test generation to produce a test suite that fulfils the test policy defined for the SUT. These criteria are then transformed (**Fig. 1**.(3)) into a test cases specification (i.e. a high-level description of a desired test case). Utting et al. [28] use the example of test case specification using state coverage of a finite state machine (FSM). In such case, a set of test case specification {reach s0, reach s1, reach s2…} where s0, s1, s2 are all the states of the FSM is the test case specification.

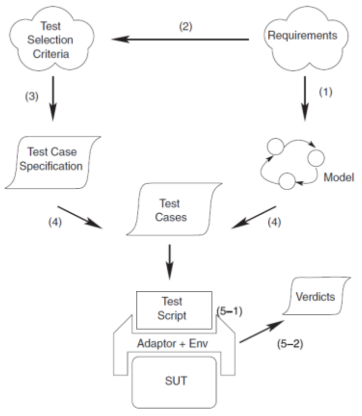


Fig. 1. The process of Model-Based-Testing (from [28])

Once the model and the test case specifications are defined, a set of test cases is generated with the aim of satisfying all the test case specifications (**Fig. 1**.(4)). With the test suite generated, the test cases are executed (either manually -i.e. by a physical person- or automatically thanks to a test execution environment). This requires concretizing the test inputs (**Fig. 1**.(5-1)) and comparing the results against expected ones to produces a verdict (**Fig. 1**.(5-2)).

2.2 Model-Based Testing of WIMP Application

In software engineering, the nearly three-decades-old field [1] that addresses concerns regarding the testing of user interfaces is called "GUI testing". In [1] GUI testing is defined as performing sequences of events (e.g., "click on button", "enter text", "open menu") on GUI widgets (e.g., "button", "text-field", "pull-down menu"). For each sequence, the test oracle checks the correctness of the state of the GUI either after each event or at the end of the sequence. Since the domain is three-decade-old, it naturally focused on WIMP UIs as they were the only available at the time. This focusing is still quite present today.

Some of the research works presented in the following paragraphs do not follow the process of MBT presented by Utting [28], but they propose relevant and inspiring approach for WIMP application testing.

Memon et al. [17] propose a detailed taxonomy of the Model-Based techniques employed to generate test cases in GUI testing. These techniques rely on various kinds of models (state machine, workflow, etc.) that target mono-event-based systems (i.e. systems on which UI events are produced directly as a result of a single action on a widget: key typed, mouse clicked, etc.). They describe the possible test cases by checking reachability of a node. It is important to mention that most of the techniques listed in [17] rely on models built by reverse engineering of the SUT [24].

Another approach based on reverse engineering is the one of Morgado et al. [19] in the iMPAcT tool. This tool uses patterns of common behavior on Android applications to automatically test them. The tool explores the SUT checking for UI patterns using a reverse engineering process. Each UI pattern has a corresponding testing strategy (a Test Pattern) that the tool applies.

Bowen et al. [7] adopt the test-first development approach in which abstract tests are built from formal specification of the system functionality (given using Z) and from a presentation model describing the interactive components (widgets) of the user interface. These abstract test cases are used to produce JUnit and UISpec4J[1] test cases.

Finally, Campos et al. [8] propose an example of approach that matches the outlines of the MBT-process by using task models to perform scenario-based testing of user-interfaces coded in Java using the Synergistic IDE Toucan [16]. The conformance between the application code and the task models is checked at runtime thanks to annotations in the Java code that allow the association of methods calls to the Interactive Input and Output Tasks. The scenarios produced from the task model are then played automatically on the Java application.

2.3 Model-Based Testing of post-WIMP Application

Testing post-WIMP applications requires going beyond GUI testing as mentioned by Lelli et al. [15]. This requires considering ad-hoc widgets and complex interaction techniques that cannot be performed simply as sequences of events on GUI widgets. For instance, interactions such as gesture-based or voice command activations are not tied to a specific GUI widget.

One of the main references in post-WIMP application testing is Malai [15] that has been proposed as a framework to describe advanced GUI Testing. It allows to describe the interaction using Finite State Machine (FSM) with two types of end state: terminal state and aborting state. These states are dedicated to identifying whether the user completed the interaction or aborted it. The output actions associated with completing the interaction (i.e. reaching its terminal state) are described in a specific reification of tools called instruments.

However, the use of FSM limits the description of interaction techniques and should be enhanced to support:

- **The description of dynamic instantiation of physical and virtual input/output devices**: on systems with a touchscreen, the display is a physical output device and the touch layer the physical input device. When dealing with multi-touch interaction, a finger is a virtual device that is added/removed whenever it touches the screen or is removed from it;

[1] https://github.com/UISpec4J/UISpec4J

- **The analysis of qualitative temporal aspect** to verify that the description supports functional requirements expressed in temporal logic (e.g. whenever the play button is enabled, the stop button is disabled);
- **The description of dynamic user interface behavior** such as animations during transition between states of the system;
- **The system configuration** as for instance, using resolution scaling on displays with high pixels densities affects the size, location and translation of the GUI elements on screen. Beyond, this also applies to mobile and web-based UI in which having a responsive-design behaving properly is a concern.

While advances have been made in the description of such aspect, especially in work such as [14], there are not, to the best of our knowledge, techniques taking advantages of them to generate tests cases for interactive applications. In the following of the paper, we introduce and use the ICO formalism to demonstrate the need for advanced modelling techniques for effective testing of interactive applications.

3 Modelling of a Post-WIMP Case Study Using ICO

In this section, we present an architecture for post-WIMP applications and highlight where the interaction techniques take place. We then present the informal requirements for the "finger clustering" interaction technique used as a case study in the remaining of this paper. Thereafter, we introduce the formal description technique we use, ICO [20], and present the models associated to the "finger clustering" interaction technique.

3.1 Architecture of a Post-WIMP Application

Effectively testing an interactive application requires a good understanding of its architecture and of the role of its components to select appropriate test criteria [9]. While a detailed architecture such as MIODMIT [10] is able to describe in detail the hardware and software components of interactive systems, we use in this paper a simpler software architecture (inspired by ARCH [3]) for touch applications, presented in **Fig. 2**, to detail the role of the component we focus on. The work presented in the remaining of this paper is still applicable to a more complex architecture.

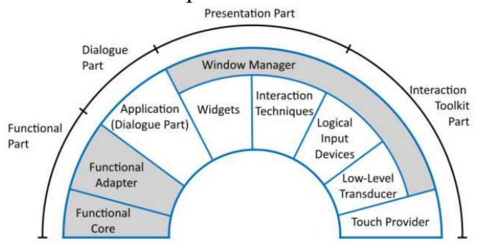


Fig. 2. Example of architecture of a post-WIMP application [13].

As this paper discusses specific aspects of post-WIMP application, we do not detail the "back-end" of the application, or Functional Part (leftmost part of **Fig. 2**). The Dialogue Part of the application shares a common role in WIMP and post-WIMP applications, i.e. translating high-level events resulting of the user interaction into invocations on the Functional Part. The main difference between WIMP and post-WIMP applications then resides in the Window Manager that contains, from right to left, the widgets (that share similar roles to widgets of WIMP interfaces), the Interaction Technique, the Logical Input Device and the Low-Level Transducer.

The Low-Level Transducer is connected to the Touch Provider (rightmost part of **Fig. 2**), i.e. the driver of the touch screen. The Touch Provider produces the lowest-level events in the input chain as they are directly derived from the touch screen behavior. The role of the Low-Level Transducer is to handle these low-level events and to translate them to make sense for the Window Manager logic. On touch applications, the Low-Level Transducer creates Logical Input Devices (i.e. Fingers) with unique IDs and additional information (coordinates, pressure level, etc.). The Logical Input Devices are added to the Window Manager Interaction Technique(s) that will notify widgets and other subscribers (such as a drawing panel) using high-level events when either simple (e.g. tap) or complex (e.g. pinch) interactions are performed.

While this paper focus on the testing of the Interaction Technique, i.e. on verifying that for a set of Logical Input Device actions, the correct high-level events are produced, we highlight the applicability of our methods to the other components of the architecture and on integration testing of these components.

3.2 Presentation of the "Finger Clustering" Interaction Technique

The case study we use in this paper is a multi-touch interaction technique that produces events when fingers are clustered (i.e. within a given range of each other) and de-clustered according to the requirements presented below. These requirements are the inputs for the MBT Process (top-right of **Fig. 1**):

- Clusters may either contain two or three fingers;
- Clusters of three fingers are always created in priority over clusters of two fingers (i.e. if 4 fingers are on the screen in a range suitable for creating a cluster of 3 fingers, a three finger cluster will be created with a finger left alone; in no occasion such circumstance may lead to the creation of two clusters of two fingers);
- The distance between two fingers must be under 100 pixels to create a 2 finger clusters;
- Clusters of three fingers are created when three fingers on the screen form a triangle with each of its edges measuring less than 100 pixels. If it happens that two fingers of an existing cluster of 2 fingers can be part of a three fingers cluster, then the three fingers cluster is created, removing the 2 fingers cluster.
- Clusters of 2 fingers are de-clustered whenever the distance between the 2 fingers it contains goes over 150 pixels;
- Clusters of 3 fingers are never de-clustered because of the length of the edges of the triangle;

- Clusters of 3 fingers are automatically de-clustered after 5 seconds;
- All the clusters cease to exist, producing the corresponding de-clustering event, whenever a finger contained in this cluster is removed from the screen.

The events produced by this interaction technique are the following ones: *twoFingersClustered*, *twoFingersDeclustered*, *threeFingersClustered*, *threeFingersDeclustered*.

3.3 ICO: A Formal Description Technique Dedicated to the Specification of Interactive Systems

The ICO formalism is a formal description technique dedicated to the specification of interactive systems [20]. It uses concepts borrowed from the object-oriented approach (dynamic instantiation, classification, encapsulation, inheritance and client/server relationship) to describe the structural or static aspects of systems and uses high-level Petri nets to describe their dynamic or behavioral aspects.

ICOs are dedicated to the modeling and the implementation of event-driven interfaces, using several communicating objects to model the system, where both the behavior of objects and the communication protocol between objects are described by the Petri net dialect called Cooperative Objects (CO). In the ICO formalism, an object is an entity featuring four components: a cooperative object which describes the behavior of the object, a presentation part (i.e. the graphical interface), and two functions (the activation function and the rendering function) which make the link between the cooperative object and the presentation part.

An ICO specification fully describes the potential interactions that users may have with the application. The specification encompasses both the "input" aspects of the interaction (i.e. how user actions affects the inner state of the application, and which actions are enabled at any given time) and its "output" aspects (i.e. when and how the application displays information relevant to the user).

This formal specification technique has already been applied in the field of Air Traffic Control interactive applications [20], space command and control ground systems [21], interactive military [5] or civil cockpits [2].

The ICO notation is fully supported by a CASE tool called PetShop [4][22]. All the models presented in the following of this paper have been edited using it. Beyond, the presented test generation techniques are part of an effort to support MBT in PetShop.

3.4 Modeling of the Interaction Technique Using ICO

Based on the requirements provided in section 3.1, we can build a model of the interaction technique (step 1 of the MBT process) using ICO. **Fig. 3** presents this model that is made of places (oval shapes), transitions (rectangular shapes) and arcs. Two communication means are proposed by ICO: a unicast and synchronous communication, represented by method calls, and a multicast asynchronous communication, represented by event handling:

- When an ICO proposes method calls, they are each mapped into a set of three places representing three communication ports (the service input, output and exception ports). For instance, on the left part of **Fig. 3**, the places called SIP_addFinger, SOP_addFinger and SEP_addFinger are the input, output and exception ports of the method addFinger. When this method is called (for instance, by the middle right transition of **Fig. 4**), a token is created, holding the parameters of the invocation and is put in place SIP_addFinger. The transitions that invoke such methods have got a 'I' on the right part of their header.
- When an ICO is able to handle events, it uses special transitions called event handlers such as transition updateFingerX in the middle of **Fig. 3**. Such transitions are described using a set of information holding the event source, the event name, extra event parameters and a condition that concerns the event parameters. In the example of transition updateFingerX, the event source is fx, a value held by place FINGERS_MERGED_BY_TWO, the event name is touchevent_up, the event parameters contain an object called info and there is no condition on the parameter. These event handlers may handle events from outer sources or from other models. When the event source is another model, this model contains transitions that raise events. Events are raised using the keyword raiseEvent in the code part of the transition and an "E->" is put in the right part of the header of the transition (see transition merge2Fingers near the middle of **Fig. 3**).

The model illustrated by **Fig. 3** represents the behavior of the "Finger Clustering" Interaction Technique described in section 3.2. This behavior may be divided into two different parts according to their role:

- **Managing fingers life cycle:** Each finger is added or removed from the interaction technique model. In between, their coordinates may be updated (i.e. the finger has moved):
  - Adding finger to the interaction technique is done using the method addFinger, implemented using the "SIP_addFinger" place, "addFinger" transition and "SOP_addFinger" place (see **Fig. 3**). This method is called by a transition of the Low-Level transducer model (see **Fig. 4**). This invocation is made each time a Finger is created to add it to the interaction technique. When the finger enters the interaction technique, it is placed in the "SINGLE FINGERS" place. This mechanism allows for dynamic appearance of fingers in the interaction technique. To ease the rest of the discussion, we limited the number of fingers instantiated in the interaction technique to 4 using the place "FINGER_LIMIT". Removing this place would remove this restriction.
  - Removing or updating fingers coordinates is performed by handling events that comes from the Low-Level transducer model (see **Fig. 4**). When a touchEvent_up is received, the corresponding finger is removed from the interaction technique model (this is the case for instance with transition remove1 on the left part of **Fig. 3**). When a touchEvent_update is received, the corresponding point (associated with a finger) is updated (this is the case for instance with transition updating1Finger on the top left part of **Fig. 3**).

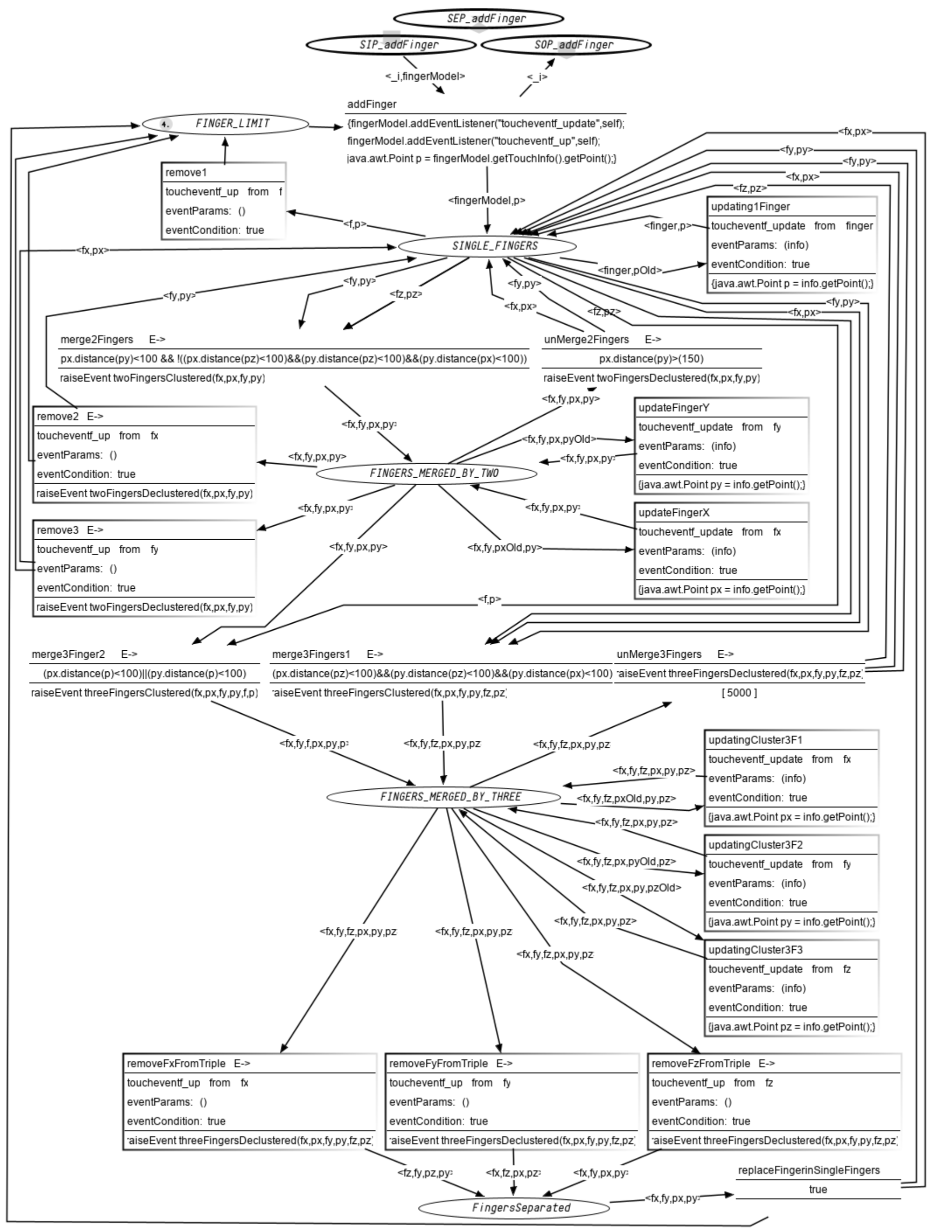

SEP_addFinger
SIP_addFinger
SOP_addFinger
<_i,fingerModel>
<_i>
addFinger
{fingerModel.addEventListener("toucheventf_update",self);
fingerModel.addEventListener("toucheventf_up",self);
java.awt.Point p = fingerModel.getTouchInfo().getPoint();}
FINGER_LIMIT
remove1
toucheventf_up from f
eventParams: ()
eventCondition: true
<fingerModel,p>
<f,p>
SINGLE_FINGERS
updating1Finger
toucheventf_update from finger
eventParams: (info)
eventCondition: true
{java.awt.Point p = info.getPoint();}
<finger,p>
<finger,pOld>
merge2Fingers E->
px.distance(py)<100 && !((px.distance(pz)<100)&&(py.distance(pz)<100)&&(py.distance(px)<100))
raiseEvent twoFingersClustered(fx,px,fy,py)
unMerge2Fingers E->
px.distance(py)>(150)
raiseEvent twoFingersDeclustered(fx,px,fy,py)
remove2 E->
toucheventf_up from fx
eventParams: ()
eventCondition: true
raiseEvent twoFingersDeclustered(fx,px,fy,py)
FINGERS_MERGED_BY_TWO
updateFingerY
toucheventf_update from fy
eventParams: (info)
eventCondition: true
{java.awt.Point py = info.getPoint();}
remove3 E->
toucheventf_up from fy
eventParams: ()
eventCondition: true
raiseEvent twoFingersDeclustered(fx,px,fy,py)
updateFingerX
toucheventf_update from fx
eventParams: (info)
eventCondition: true
{java.awt.Point px = info.getPoint();}
merge3Finger2 E->
(px.distance(p)<100)||(py.distance(p)<100)
raiseEvent threeFingersClustered(fx,px,fy,py,f,p)
merge3Fingers1 E->
(px.distance(pz)<100)&&(py.distance(pz)<100)&&(py.distance(px)<100)
raiseEvent threeFingersClustered(fx,px,fy,py,fz,pz)
unMerge3Fingers E->
raiseEvent threeFingersDeclustered(fx,px,fy,py,fz,pz)
[5000]
FINGERS_MERGED_BY_THREE
updatingCluster3F1
toucheventf_update from fx
eventParams: (info)
eventCondition: true
{java.awt.Point px = info.getPoint();}
updatingCluster3F2
toucheventf_update from fy
eventParams: (info)
eventCondition: true
{java.awt.Point py = info.getPoint();}
updatingCluster3F3
toucheventf_update from fz
eventParams: (info)
eventCondition: true
{java.awt.Point pz = info.getPoint();}
removeFxFromTriple E->
toucheventf_up from fx
eventParams: ()
eventCondition: true
raiseEvent threeFingersDeclustered(fx,px,fy,py,fz,pz)
removeFyFromTriple E->
toucheventf_up from fy
eventParams: ()
eventCondition: true
raiseEvent threeFingersDeclustered(fx,px,fy,py,fz,pz)
removeFzFromTriple E->
toucheventf_up from fz
eventParams: ()
eventCondition: true
raiseEvent threeFingersDeclustered(fx,px,fy,py,fz,pz)
replaceFingerinSingleFingers
true
FingersSeparated

Fig. 3. ICO Model for the finger clustering interaction technique.

- **Detecting clusters of fingers:** Each time a finger is added or removed from the interaction technique model, or each time the coordinates of one finger is updated, the clustering or de-clustering of fingers is computed:

- For two or three fingers, the principle is the same, supported thanks to the preconditions of the “mergeXFingersX” and “unMergeXFingers” transitions, that compute the proximity of the fingers.
- The 5 seconds timeout for de-clustering three fingers is handled thanks to a “timed transition” (note the [5000] - expressed in ms - line at the bottom of the unMerge3Fingers transition) that removes the fingers held by place FINGERS_MERGED_BY_THREE.

While we are able to describe the interaction technique, the approach can be applied to other components of the architecture. For instance, **Fig. 4** presents the ICO model of the Low-Level Transducer component of the architecture presented earlier. Note that the “addFingerToInteraction” transition contains an invocation on the interaction technique. This invocation is the one associated with the SIP/SOP places in the Interaction Technique Model. To prevent inconsistent input such as two fingers at the same location (which is physically impossible), a test arc allows to check whether a touch down is associated with a touch point of a finger already on the screen.

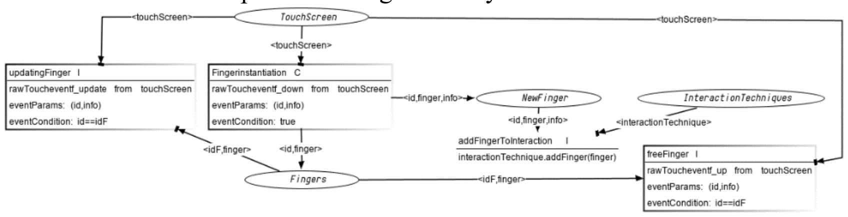


Fig. 4. ICO Model for the Low-Level Transducer

4 Generating Test Cases from ICO Specifications

In this section, we focus on steps 2, 3 and 4 of the MBT process (see **Fig. 1**) applied to our case study. We first present our test selection criteria and specification and then present our test generation approach.

4.1 Test Selection Criteria and Test Case Specification

Testing an interaction technique consists in verifying that, for a set of low-level input events, the corresponding high-level event is produced so that components subscribed to it (e.g. application dialogs or widgets) are notified with a well-formed event. This differs from testing the application as done in the work presented in section 2.2. Indeed, in these, the events considered in the test cases are already high-level ones and the verification that is made is that the effect on the UI is the correct one. To perform testing on the interaction techniques requires to i) describe the sequences of actions triggering the events raised by the interaction techniques and to ii) describe the associated events to observe on the interaction technique.

Regarding the finger clustering interaction techniques, this means that we want to be able to identify all the possible sequences of low-level events leading to the raising of the “twoFingersClustered”, “threeFingersClustered”, “twoFingersDeclustered and "threeFingersDeclustered" events in the interaction technique transitions. For illustration purpose, we focus on the raising of "threeFingersClustered" event.

4.2 Generating Test Cases for the Interaction Technique

To identify the relevant test cases for the raising of the "threeFingersClustered" event, we use the reachability graph of the Petri-net. A reachability graph of a Petri-net is a directed graph G=(V,E), where v∈V represents a class of reachable markings; e∈E represents a directed arc from a class of markings to the other class of markings [30]. **Fig. 5** presents the reachability graph of the interaction technique introduced previously. In this graph, each state contains four digits symbolizing the number of tokens contained in the places "FINGER LIMIT", "SINGLE FINGER", "FINGERS MERGED BY TWO" and "FINGERS MERGED BY THREE". For instance, the state "4,0,0,0" at the top means that the "FINGER LIMIT" place contains 4 tokens and that the other places are empty. We take advantage of the APT (Analysis of Petri nets and labelled transition systems) project[2] [6] to generate this graph.

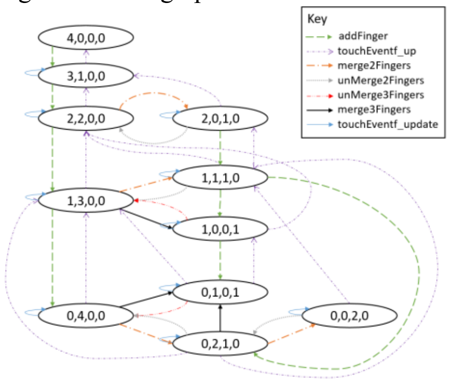


Fig. 5. Reachability graph derived from the ICO model of the interaction technique

As observable in **Fig. 5**, the reachability graph is actually a Finite State Machine with no accepting state. Considering that the event we focus on is raised in the "merge3FingersX" transition, we know that the event must be raised whenever a state of the FSM having a "merge3Fingers" incoming edge is reached. Marking these states (i.e. "1,0,0,1" and "0,1,0,1") as accepting ones allows us to describe the actual grammar of the test cases for the "threeFingersClustered" event. This grammar[3] only misses concrete values for fingers coordinates. The following is an example of test case matching this grammar expressed into Backus-Naur Form (BNF):

```
<testCase> ::= <addFinger> <touchEventf_update> <addFinger> <addFinger> <touchEventf_update> <merge3Fingers>
```

[2] https://github.com/CvO-Theory/apt

[3] For which the regular expression can be obtained from the FSM using tools such as FSM2Regex (http://ivanzuzak.info/noam/webapps/fsm2regex/)

The reachability graph we present in this case study contains values for each place as we intentionally limited to 4 the number of fingers in the interaction technique. However, some touch screens support more than 4 fingers and therefore one may want to use multiple clusters of three fingers. It would be possible to remove this restriction while still being able to apply our process by performing our analysis on a symbolic reachability graph. Symbolic reachability graphs use variables instead of concrete values in the states for the analysis of Petri-nets with such infinite marking, making it possible to express infinite number of states.

To prepare the instantiation of the test scripts, we must focus on how the required values are produced, partly supported by the model of the application. This model describes the conditions under which the transitions are fired. In our case, it describes the constraints on the distance between the points, defining the values domain. When instantiating the test scripts, the integration of these constraints relies on a semi-automated support, where the values are checked at editing time. For instance, in the instantiation of the grammar example proposed above, whatever the coordinates of the three added fingers are, the distance between them must fit the precondition of the transitions "merge3Fingers1" and "merge3Fingers2".

5 Test Cases Execution

In this section, we discuss the execution of the test of the interaction technique, i.e. steps 5.1 and 5.2 of the MBT process. While the advances we propose are mostly related to test cases generation, we find it important to emphasis the relevance of selecting the test adapter appropriately and to discuss the possible ways to use our test cases.

5.1 Test Adapter Selection

Testing the interaction technique consist in verifying that for a set of input events the corresponding high-level event is produced. A key in executing such test properly is being able to produce an input event that is actually the event expected by the interaction technique as an input, i.e. an event from the low-level transducer. Assuming that we are testing our interaction technique as part of a JavaFX application, this means producing JavaFX Touch Events[4]. However, testing the interaction technique alone may prove to be insufficient to ensure that the interaction technique will behave properly for the end-user. Indeed, while evaluating our approach, we encountered a known issue that no touch events are forwarded to JavaFX by most popular distributions of Linux using a GTK-based desktop environment[5]. In other words, the Touch Provider of these distributions is not producing relevant events for the Low-Level Transducer that cannot, in turn, produce events for the interaction technique. This means that the JavaFX finger clustering cannot be used on a Linux platform even though tests based on JavaFX Touch Event would have indicated that the interaction

[4] https://openjfx.io/javadoc/11/javafx.graphics/javafx/scene/input/class-use/TouchEvent.html
[5] https://bugs.openjdk.java.net/browse/JDK-8090954

technique behaves properly. Therefore, when testing touch applications, one may want to produce Operating System-level events and to perform integration testing of the Low-Level Transducer/Interaction Technique couple. Such tests can be executed on the Windows platform by using the Touch Injection technology of the Windows API[6] to produce OS-level touch events as inputs. Regarding Linux, it is worth mentioning that ARM versions of GTK are not prone to the issue presented earlier.

5.2 Test Execution for the Interaction Technique

The execution of the tests on the SUT is an activity that is highly dependent of the way the SUT is implemented. Overall, testing the interaction technique alone requires i) being able to forward the event sequence of the test script to the interaction technique and ii) being able to subscribe to the events the interaction technique produces. The easiest way to test the interaction technique of the SUT is to do it using white-box or grey-box testing. Indeed, in such cases, it is easy to either instrument the class of the SUT responsible for the interaction technique or to encapsulate it in a test adapter with which the test execution environment can interact. Then, the test execution environment can perform the event sequence described by the test script. The role of the oracle is then to determine whether the test passed based on whether or not it received the expected event from the interaction technique in a timely manner.

6 Generalizability of the Approach

While this paper focused on the interaction technique component of the architecture presented in section 3.1, the ICO notation, alongside with its CASE tool Petshop, support the modelling and the test generation for other components of the architecture as well as GUI Testing as defined by Banerjee et al. [1]. This section highlights the generalizability of the modelling philosophy and of the test case generation approach. Due to space constraint and to the highly SUT-dependent nature of the tests execution, we will however not develop further on test execution.

6.1 Generalizability of the Modelling Philosophy

In addition to interaction techniques, we pointed out in section 3.4 that ICO can be used to model the low-level transducer of a post-WIMP application (**Fig. 4**). Modelling of Logical Input Devices (e.g. fingers) and their dynamic instantiation is covered in [14]. Moreover, [20] demonstrates that ICO allows the description of the Application (dialog part) components, including those with dynamic instantiation of widgets, on examples such as an Air Traffic Control (ATC) plane manager. To validate that our work is compatible with GUI Testing of WIMP application, we modelled the application specified in Memon et al.'s [17] review of advances in MBT for applications with a GUI front-end. We had no trouble describing the behavior of this WIMP application using ICO in

[6] https://docs.microsoft.com/en-us/windows/desktop/api/_input_touchinjection/

Petshop. Combining this with the modelling of post-WIMP interaction techniques demonstrated herein, shows that we are able to model post-WIMP applications.

6.2 Generalizability of the Test Case Generation Approach

Thanks to Memon et al.'s review of advances in MBT [17], we were able to verify that our test generation approach worked for WIMP applications. Indeed, [17] presented various models for the application it specifies, including one being a Finite State Machine. This allowed us to verify that the reachability graph of the Petri-net was the same (name of states aside) as the FSM in [17]. Beyond that, on applications that involve dynamicity such as the ATC plane manager dialog, the approach fits well as each aircraft is added to the dialog model using invocation in the same way as fingers are added to the interaction technique presented in this paper. Yet, as the number of aircraft on the radar visualization is virtually infinite, the use of a symbolic reachability graph is made mandatory, while standard reachability graph can be kept for interaction techniques (as the maximum number of touch points supported by the screen is known).

7 Conclusion and Future Work

Testing interactive applications is known to be a challenging activity, whether we consider WIMP or post-WIMP applications. In this paper, we have shown that while the testing of WIMP applications retained most of the attention of researchers and practitioners in the field of MBT, post-WIMP applications raise new challenges for the community. Indeed, properly testing post-WIMP following the standard Model-Based Testing process requires modelling techniques that are expressive enough to describe the dynamic instantiation of virtual and physical devices, temporal aspects, system configuration, etc. Only such models allow the generation of exhaustive enough test cases.

Building on previous work on the Petri-net-based notation ICO (and its associated CASE tool, PetShop), we showed that we are able to propose a toolchain that addresses the need for expressive modelling techniques in order to support the generation of test cases for post-WIMP application following the MBT process. We showed that thanks to the mechanism supported by ICO we are able to support the high dynamicity of post-WIMP applications for all the software components of the architecture. This expressiveness allows for the generation of abstract test case using a grammar derived from the reachability graph of Petri-nets.

As we focused on a specific component of the architecture, i.e. the interaction technique, we found that post-WIMP applications are more sensitive than WIMP applications to the execution platform, as touch event are not always well forwarded to libraries by operating-systems, highlighting the need for integration testing. A future extension to our work would be to implement the generation of integration test cases into PetShop by relying on the different artifacts allowing the communication between models.

Finally, we are currently investigating using such approach for the testing of interactive applications to be deployed in large civil aircraft interactive cockpits. Indeed, following guidance from supplement DO-333 [26] on formal methods to the DO-178C certification process [25], one may use formal specifications during the development of such application. If a formal model of the interactive application is built for supporting reliability arguments (e.g. “low-level requirements are accurate and consistent [25]”) we propose to exploit that model to generate test cases from that formal specification (as proposed by Gaudel [11]). Such process could result in more cost-effective test case generation leveraging on available formal models. Beyond, thanks to the expressive power of ICO, such approach could support the adoption of application offering richer interaction techniques (e.g. animations [18] or multitouch [12]) even in safety-critical context (e.g. brace touch [23]).

References

1. Banerjee, I., Nguyen, B., Garousi, V., Memon, A.M.: Graphical user interface (GUI) testing: Systematic mapping and repository. Information and Software Technology. 55, 1679–1694 (2013).
2. Barboni E., Conversy S., Navarre D. & Palanque P. Model-Based Engineering of Widgets, User Applications and Servers Compliant with ARINC 661 Specification. 13th conf. on Design Specification and Verification of Interactive Systems (DSVIS 2006), LNCS Springer Verlag. p25-38
3. Bass, L., Little, R., Pellegrino, R., Reed, S., Seacord, R., Sheppard, S. and Szezur, M.R. The arch model: Seeheim revisited. In User Interface Developpers' Workshop (1991).
4. Bastide, R., Navarre, D., Palanque, P.: A Model-based Tool for Interactive Prototyping of Highly Interactive Applications. In: CHI ’02 Extended Abstracts on Human Factors in Computing Systems. pp. 516–517. ACM, New York, NY, USA (2002).
5. Bastide R., Navarre D., Palanque P., Schyn A. & Dragicevic P. A Model-Based Approach for Real-Time Embedded Multimodal Systems in Military Aircrafts. Sixth International Conference on Multimodal Interfaces (ICMI'04) October 14-15, 2004, USA, ACM Press.
6. Best, E., Schlachter, U.: Analysis of Petri Nets and Transition Systems. Electron. Proc. Theor. Comput. Sci. 189, 53–67 (2015).
7. Bowen, J., Reeves, S.: Generating Obligations, Assertions and Tests from UI Models. Proc. ACM Hum.-Comput. Interact. 1, 5:1–5:18 (2017).
8. Campos, J.C., Fayollas, C., Martinie, C., Navarre, D., Palanque, P., Pinto, M.: Systematic Automation of Scenario-based Testing of User Interfaces. In: Proceedings of the 8th ACM SIGCHI Symposium on Engineering Interactive Computing Systems. pp. 138–148. ACM, New York, NY, USA (2016).
9. Canny, A., Bouzekri, E., Martinie, C., Palanque, P.: Rationalizing the Need of Architecture-Driven Testing of Interactive Systems. In: Human-Centered and Error-Resilient Systems Development. Springer, Cham (2018).
10. Cronel, M., Dumas, B., Palanque, P., Canny, A.: MIODMIT: A Generic Architecture for Dynamic Multimodal Interactive Systems. In: Bogdan, C., Kuusinen, K., Lárusdóttir, M.K., Palanque, P., and Winckler, M. (eds.) Human-Centered Software Engineering. pp. 109–129. Springer International Publishing (2019).
11. Gaudel, M.-C.: Testing can be formal, too. In: TAPSOFT ’95: Theory and Practice of Software Development. pp. 82–96. Springer, Berlin, Heidelberg (1995).
12. Hamon, A., Palanque, P., Silva, J.L., Deleris, Y., Barboni, E.: Formal Description of Multi-touch Interactions. In: Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems. pp. 207–216. ACM, New York, NY, USA (2013).

13. Hamon-Keromen, A.: Définition d'un langage et d'une méthode pour la description et la spécification d'IHM post-W.I.M.P. pour les cockpits interactifs, (2014).
14. Hamon, A., Palanque, P., Cronel, M., André, R., Barboni, E., Navarre, D.: Formal Modelling of Dynamic Instantiation of Input Devices and Interaction Techniques: Application to Multi-touch Interactions. In: Proceedings of the 2014 ACM SIGCHI Symposium on Engineering Interactive Computing Systems. pp. 173–178. ACM, New York, NY, USA (2014).
15. Lelli, V., Blouin, A., Baudry, B., Coulon, F.: On model-based testing advanced GUIs. In: 2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW). pp. 1–10 (2015).
16. Martinie, C., Navarre, D., Palanque, P., Barboni, E., Canny, A.: TOUCAN: An IDE Supporting the Development of Effective Interactive Java Applications. In: Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems. pp. 4:1–4:7. ACM, New York, NY, USA (2018).
17. Memon, A.M., Nguyen, B.N.: Advances in Automated Model-Based System Testing of Software Applications with a GUI Front-End. In: Zelkowitz, M.V. (ed.) Advances in Computers. pp. 121–162. Elsevier (2010).
18. Mirlacher, T., Palanque, P., Bernhaupt, R.: Engineering Animations in User Interfaces. In: Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems. pp. 111–120. ACM, New York, NY, USA (2012).
19. Morgado, I.C., Paiva, A.C.R.: The iMPAcT Tool for Android Testing. Proc. ACM Hum.-Comput. Interact. 3, 4:1–4:23 (2019).
20. Navarre, D., Palanque, P., Ladry, J.-F., Barboni, E.: ICOs: A Model-based User Interface Description Technique Dedicated to Interactive Systems Addressing Usability, Reliability and Scalability. ACM Trans. Comput.-Hum. Interact. 16, 18:1–18:56 (2009).
21. Palanque P., Bernhaupt R., Navarre D., Ould M. & Winckler M. Supporting Usability Evaluation of Multimodal Man-Machine Interfaces for Space Ground Segment Applications Using Petri net Based Formal Specification. Ninth Int. Conference on Space Operations, Italy, June 18-22, 2006.
22. Palanque P., Ladry J-F, Navarre D. & Barboni E. High-Fidelity Prototyping of Interactive Systems can be Formal too 13th Int. Conf. on Human-Computer Interaction (HCI International 2009) LNCS, Springer.
23. Palanque P., Cockburn A., Gutwin C., Deleris Y. & Desert-Legendre L. Brace Touch: A Dependable, Turbulence-Tolerant, Multi-Touch Interaction Technique for Interactive Cockpits. In: International Conference on Computer Safety, Reliability, and Security 2019 (SAFECOMP). Springer, Verlag (2019).
24. Pezzè, M., Rondena, P., Zuddas, D.: Automatic GUI Testing of Desktop Applications: An Empirical Assessment of the State of the Art. In: Companion Proceedings for the ISSTA/ECOOP 2018 Workshops. pp. 54–62. ACM, New York, NY, USA (2018).
25. RTCA. DO-178C Software Considerations in Airborne Systems and Equipment Certification. 2011.
26. RTCA. DO-333 Formal Methods Supplement to DO-178C and DO-278A. 2011.
27. Shneiderman, B.: Direct Manipulation: A Step Beyond Programming Languages. Computer. 16, 57–69 (1983). https://doi.org/10.1109/MC.1983.1654471.
28. Utting, M., Pretschner, A., Legeard, B.: A taxonomy of model-based testing approaches. Softw. Test. Verif. Reliab. 22, 297–312 (2012).
29. Van Dam, A.: Post-WIMP user interfaces. Communications of the ACM. 40.2, 63-67 (1997).
30. Ye, X., Zhou, J., Song, X.: On reachability graphs of Petri nets. Computers & Electrical Engineering. 29, 263–272 (2003). https://doi.org/10.1016/S0045-7906(01)00034-9.

Modelling Human Reasoning in Practical Behavioural Contexts using Real-time Maude[⋆]

Antonio Cerone[1] and Peter Csaba Ölveczky[2]

[1] Department of Computer Science, Nazarbayev University, Nur-Sultan, Kazakhstan
antonio.cerone@nu.edu.kz
[2] Department of Informatics, University of Oslo, Norway
peterol@ifi.uio.no

Abstract. In this paper we present an approach for modelling human reasoning using rewrite systems and we illustrate our approach in the context of human behaviour through a car driving example. Reasoning inference rules and descriptions of human activities are expressed using the Behavior and Reasoning Description Language (BRDL). The BRDL model is then translated into Real-time Maude. The object-oriented and equational logic aspects of Maude are exploited in order to define alternative semantic variations of BRDL that implement alternative theories of memory and cognition.

Keywords: Human Reasoning · Human Behaviour · Formal Methods · Rewrite Systems · Real-time Maude.

1 Introduction

One of the main challenges in Human-computer Interaction (HCI) is that the way humans actually use devices is not always consistent with the use for which such devices have been designed and built. In fact, although a systematic exploration of the concept of "plausible" behaviour may provide a good baseline for understanding the interaction [3, 9], some forms of "plausible" behaviour emerge only in specific contexts and cannot be predicted *a priori*. Cognitive architectures [10], formal methods [17, 18] and several other approaches, including machine learning and control theory [17], have been used to tackle this problem.

However, cognitive architectures tend to be specialised, each with a specific scope, which is normally academic and seldom practical [10], formal methods are "regarded as requiring too much expertise and effort for day-to-day use, being principally applied in safety-critical areas outside academia" [17, Ch. 7, page 187], and machine learning and control theory focus on the interaction process rather than human behaviour. Moreover, although emulating reasoning is one of the main objectives of some cognitive architectures, past and current efforts in this sense either do not consider human errors or are detached from

⋆ Work partly funded by Seed Funding Grant, Project SFG 1447 "Formal Analysis and Verification of Accidents", University of Geneve, Switzerland.

the practical context of human behaviour [10]. Furthermore, high-level reasoning is not supported by control theory and, although it may emerge using machine learning, the way it emerges cannot be explained.

Our approach builds on the *Behaviour and Reasoning Description Language (BRDL)* [8] and on the use of the Maude rewrite system [14–16] to model the dynamics of human memory and memory processes [6, 7]. The semantics of BRDL is based on a basic model of human memory and memory processes and is adaptable to different cognitive theories. This allows us, on the one hand, to keep the syntax of the language to a minimum, thus making it easy to learn and understand without requiring expertise in mathematics or formal methods and, on the other hand, to use alternative semantic variations to compare alternative theories of memory and cognition. BRDL, is equipped with the linguistic constructs to specify reasoning goals, inference rules and unsolved problems. We use rewrite systems [12, 15] to implement such constructs. Specifically, BRDL is translated into Real-time Maude [14, 16], thus combining human components with the system components that model the environment in which humans operate.

The rest of the paper is structured as follows. Sections 2 and 3 present overviews of Real-time Maude and BRDL, respectively. Section 4 presents the Real-time Maude implementation of the model of human memory and memory processes that provide the dynamics of BRDL constructs. Section 5 illustrates the rewrite rules to emulate human reasoning and the environment in which humans operate. Finally, Section 6 concludes the paper.

2 Real-Time Maude

Real-Time Maude [14, 16] is a formal modeling language and high-performance simulation and model checking tool for distributed real-time systems. It is based on Full Maude, the object-oriented extension of Core Maude, which is the basic version of Maude.

An algebraic equational specification (specifying sorts, subsorts, functions and equations defining the functions) defines the data types in a functional programming style. Labeled rewrite rules `crl [`l`]: ` t `=>` t' `if` *cond* define local transitions from state t to state t', and tick rewrite rules `crl [`l`]: {`t`} => {`t'`} in time` Δ `if` *cond* advance time in the *entire* state t by Δ time units.

We briefly summarize the syntax of Real-Time Maude and refer to Ölveczky's work [14–16] for more details. Maude *equational logic* supports declaration of *sorts*, with keyword `sort` for one sort, or `sorts` for many. A sort `A` may be specified as a subsort of a sort `B` by `subsort A < B`. Operators are introduced with the `op` and `ops` keywords: `op` f `:` $s_1 \dots s_n$ `->` s. They can have user-definable syntax, with underbars '`_`' marking the argument positions. Some operators can have equational *attributes*, such as `assoc`, `comm`, and `id`, stating that the operator is associative, commutative and has a certain identity element, respectively. Such attributes are used by the Maude engine to match terms *modulo* the declared axioms. An operator can also be declared to be a constructor (`ctor`) that defines the carrier of a sort. Equations and rewrite rules are introduced with, respectively, keywords `eq`, or `ceq` for conditional equations, and `rl`, or `crl` for conditional rules. The mathematical variables in such statements are declared with the keywords `var` and `vars`, or can be introduced on the fly in a statement without being declared previously, in which case they have the form *var*`:`*sort*. An equation $f(t_1, \ldots, t_n) = t$ with the `owise` (for "otherwise") attribute can be applied to a subterm $f(\ldots)$ only if no other equation with left-hand side $f(u_1, \ldots, u_n)$ can be applied.

A declaration `class` C `|` att_1 : s_1, ..., att_n : s_n declares a class C with attributes att_1 to att_n of sorts s_1 to s_n. An *object* of class C is represented as a term `<` O : C | att_1 : $val_1, ..., att_n$: val_n `>` of sort `Object`, where O, of sort `Oid`, is the object's *identifier*, and where val_1 to val_n are the current values of the attributes att_1 to att_n. The state is a term of sort `Configuration`, and is a *multiset* of objects and messages. Multiset union is denoted by an associative and commutative juxtaposition operator, so that rewriting is *multiset rewriting.*

Real-Time Maude specifications are executable, and the tool provides a variety of formal analysis methods. The *timed rewriting* command `(tfrew` t `in time <=` *timeLimit* `.)` simulates *one* of the system behaviors by rewriting the initial state t up to duration *timeLimit*.

3 Behavior and Reasoning Description Language (BRDL)

The Behavior and Reasoning Description Language (BRDL) [8] originates from and extends the *Human Behaviour Description Language (HBDL)* introduced in previous work [6, 7]. HBDL aims at the modelling of automatic and deliberate human behaviour while interacting with an environment consisting of heterogenous physical components. It requires reasoning and problem solving aspects to be modelled explicitly in a procedural way, whereby the reasoning process and the problem solution are explicitly described with the language. BRDL, instead, is equipped with the linguistic constructs to specify reasoning goals, inference rules and unsolved problems. It is then the cognitive engine that implements the language to emulate the reasoning and problem solving processes.

BRDL is based on Atkinson and Shiffrin's *multistore model* of human memory [1]. This model is characterised by three stores between which various forms of information flow: *short-term memory (STM)*, which has a limited capacity and where the information that is needed for processing activities is temporarily stored with rapid access and rapid decay, and *long-term memory (LTM)*, which has a virtually unlimited capacity and where information is organised in structured ways, with slow access but little or no decay. A usual practice to keep information in memory is *rehearsal.* In particular, *maintenance rehearsal* allows us to extend the time during which information is kept in STM, whereas *elaborative rehearsal* allows us to transfer information from STM to LTM [2]. We consider a further decomposition of LTM: *semantic memory*, which refers to our *knowledge* of the world and consists of the *facts* that can be *consciously* recalled, and *procedural memory*, which refers to our *skills* and consists of *rules* and *procedures* that we *unconsciously* use to carry out tasks, particularly at the motor level.

BRDL has a concise, appealing syntax, which is presented elsewhere [8]. In order to show how BRDL is translated to Maude, in this section we introduce an ASCII, verbose version of the syntax, as it is implemented in Maude. Both HDBL and BRDL describe human behaviour through the manipulation of three kinds of entities:

perceptions are sensed in the environment and enter human input channels;
actions are performed by the human on the environment;
cognitive information consists in the items we store in our STM, including information retrieved from the LTM, goals, recent perceptions or planned actions.

3.1 BRDL Entities and Cognitive Control

Pieces of cognitive information are also components of the associations in semantic memory and the procedures in procedural memory. Such entities are modelled with Maude using the following sort structure.

```
sorts Perception Action Cognition BasicItem Item Goal .
subsorts Cognition Perception Action < BasicItem < Item .
subsort Goal < Item .
```

where `Perception`, `Action` and `Cognition` model perceptions, actions and cognitive information, respectively. Sort `Item` models anything that can be stored in STM and sort `BasicItem` is its subsort that excludes goals (from sort `Goal`). All these entities may also be elements of sets that define further sorts as follow.

```
subsorts Perception < PerceptionSet < BasicItemSet .
subsorts Cognition < CognitionSet < BasicItemSet .
subsorts Action < ActionSet < BasicItemSet .
subsort BasicItem < BasicItemSet .
subsorts EmptyItemSet < PerceptionSet CognitionSet ActionSet
                      < BasicItemSet < ItemSet .
subsort Item < ItemSet .
op none : -> EmptyItemSet [ctor] .
op _;_ : BasicItemSet BasicItemSet ->
                 BasicItemSet [ctor assoc comm id: none] .
op _;_ : PerceptionSet PerceptionSet ->
                 PerceptionSet [ctor assoc comm id: none] .
op _;_ : ActionSet ActionSet ->
                  ActionSet [ctor assoc comm id: none] .
op _;_ : ItemSet ItemSet -> ItemSet [ctor ditto] .
```

We use semicolon "`;`" as the general operator to add elements or subsets to a set, starting from an empty set (`none` in this case).

We extend `Perception` to `DefPerception` and `Action` to `DefAction` by including as default values `noPerception` and `noAction` to model the absence of perception and action, respectively.

```
sorts DefPerception DefAction .
subsort Perception < DefPerception . subsort Action < DefAction .
op noAction : -> DefAction [ctor] .
op noPerception : -> DefPerception [ctor] .
```

Only relevant perceptions are transferred, possibly after some kind of processing, to the STM using *attention*, a selective processing activity that aims to focus on one aspect of the environment while ignoring others. *Explicit attention* is associated with our goal in performing a task. It focusses on goal-relevant stimuli in the environment. *Implicit attention* is grabbed by sudden stimuli that are associated with the current mental state or carry emotional significance. Inspired by Norman and Shallice [13], we consider two levels of cognitive control:

automatic control
: fast processing activity that requires only *implicit attention* and is carried out outside awareness with no conscious effort implicitly, using rules and procedures stored in the procedural memory;

deliberate control
: processing activity triggered and focussed by *explicit attention* and carried out under the intentional control of the individual, who makes explicit use of facts and experiences stored in the declarative memory and is aware and conscious of the effort required in doing so.

In order to model automatic and deliberate control as well as reasoning, we introduce the following sorts and operations.

```
sorts Automatism KnowledgeDomain .
op automatism : KnowledgeDomain -> Automatism [ctor] .
op goal : KnowledgeDomain BasicItemSet -> Goal [ctor] .
op infer : KnowledgeDomain -> Inference [ctor] .
```

We define automatic behaviour in terms of a specific knowledge domain (sort `KnowledgeDomain` and operation `automatism`). Automatic behaviour is driven by the knowledge domain, which gives a focus to implicit attention.

Deliberate behaviour is driven by a goal, which not only depends on the knowledge domain but also on a representation of the goal achievement. This representation may be given by a combination of perceptions, actions and cognitive information. For example when we are interacting with an ATM (Automatic Teller Machine) with the goal of withdrawing cash, we achieve the goal when we perform the action of collecting the cash.

Inference is driven by the knowledge domain on which we are reasoning.

3.2 Basic Activities

Human behaviour is modelled in BRDL (and HTDL) as a set of *basic activities*, defined through the following sorts and operations

```
sorts  AutomaticActivity DeliberateActivity Knowledge .
op _:_>|_-->_|>_duration_ : Automatism BasicItemSet DefPerception
```

```
        DefAction ItemSet Time -> AutomaticActivity . [ctor]
op _:_>|_-->_|>_duration_ :Goal BasicItemSet DefPerception
        DefAction ItemSet Time -> DeliberateActivity [ctor] .
op _:_>|-->|>_duration_ : Inference BasicItemSet
        ItemSet Time -> Knowledge [ctor] .
```

An automatic basic activity within a given knowledge domain *domain* is modelled in BRDL and HTDL as

`automatism(`*domain*`) :` *info1* `>|` *perception* `-->` *action* `|>` *info2* `duration` *d*

where *info1* is the triggering cognitive information in STM, *perception* is the triggering perception, *action* is the performed action, *info2* is a new cognitive information stored in the STM, and *d* is the duration of the mental processing. Symbol "`>|`" denotes that *info1* is removed from the STM and "`|>`" denotes that *info2* is stored in the STM. Using derived operations (i.e. not defined as constructors but through equations) we have the following syntactic sugar

`automatism(`*domain*`) :` *info1* `|` *perception* `-->`
action `|>` *info2* `duration` *d*

where *info1* acts as a trigger but is not is removed from STM, and

`automatism(`*domain*`) :` *info* `|` *info1* `>|` *perception* `-->`
action`|>` *info2* `duration` *d*

where the union *info*`;`*info1* acts as a trigger but only *info1* is removed from STM.

A deliberate basic activity within a given knowledge domain *domain* is modelled in BRDL and HTDL as

`goal(`*domain*, *info*`) :` *info1* `>|` *perception* `-->` *action* `|>` *info2* `duration` *d*

where *info* is the information denoting the achievement of the goal.

An inference within a given knowledge domain *domain* is modelled in BRDL as

`inference(`*domain*`) :` *info1* `>|-->|>` *info2* `duration` *d*

where *info1* is the premise and *info2* is the consequence.

Syntactic sugar for deliberate basic activities and inferences is defined similarly to automatic basic activities.

Procedural memory is modeled as sort `ProcMem`, which is a set of automatic basic activities

```
sort ProcMem . subsort AutomaticActivity < ProcMem .
op emptyPM : -> ProcMemory [ctor] .
op _;_ : ProcMemory ProcMemory -> ProcMemory
     [ctor assoc comm id: emptyPM] .
```

Semantic memory is modeled by two sort, sort `ActivMem`, which is a set of deliberate basic activities,

```
sort ActivMem . subsort DeliberateActivity < ActivMem .
op emptyAM : ->  ActivMem [ctor] .
op _;_ :  ActivMem  ActivMem ->  ActivMem [ctor assoc comm id: emptyASM] .
```

and sort `InferMem` , which is a set of inferences,

```
sort InferMem . subsort Knowledge < InferMem .
op emptyIM : ->  InferMem [ctor] .
op _;_ :  InferMem  InferMem  ->  InferMem [ctor assoc comm id: emptyIM] .
```

3.3 'Zebra Crossing' Example

As an example to illustrate these forms of human behaviour and reasoning, let us consider car driving. The knowledge domain is given by constant operation

```
op driving : -> KnowledgeDomain [ctor] .
```

Automatic control is essential in properly driving a car and, in such a context, it develops throughout a learning process based on deliberate control. During the learning process the driver has to make a conscious effort that requires explicit attention. For example, the learner has to explicitly pay attention to the other cars, the pedestrian walking on the footpath, who may be ready to walk across the road, the presence of zebra crossings, traffic lights, road signals, etc. These are goals that drive explicit attention. Moreover, the information gathered through this process has to be deliberately used to achieve goals (deliberate control), which continuously emerge while driving as a learner.

For instance, let us define perceptions, actions and cognitive information of a driver dealing with a zebra crossing as follows

```
ops static moving ped zebra : Oid -> Perception [ctor] .
ops stop go : Action [ctor] .
ops givenWayPed waitForPed leftZebraCrossing : -> Cognition [ctor] .
```

The role of such constructors will be explained later in this section.

A learner's perception of an approaching zebra crossing, normally by seeing a road signal, either a horizontal or vertical one, triggers the storage of the cognitive representation of this perception in STM. We may model this instance of explicit attention as follows.

```
goal(driving,zebra) : none | zebra --> noAction |> zebra duration d1
```

where `zebra` denotes the perception of the zebra crossing and occurs three times to model, from left to right, the achievement of the goal of explicitly perceiving the presence of the zebra crossing, the actual perception and the representation of the perception in STM, respectively. There is no resultant action since here we are modelling attention.

Once also pedestrians ready to cross are perceived as follows

```
goal(driving,ped) : none | ped --> noAction |> ped duration d2
```

if the car is moving, and obviously the driver is (cognitively) aware of it (modelled by `moving` in the STM), and also the cognitive representations of perceptions `zebra` and `ped` are in STM, this composite mental state triggers the retrieval of the following inference, which models the road code rule concerning zebra crossings

```
inference(driving) :
     moving ; zebra ; ped |-->|> goal(driving,givenWayPed) duration d3
```

Retrieving the rule results in adding goal `goal(driving,givenWayPed)` to the STM without removing `moving`, `zebra` and `ped`. Such a goal dictates the prescribed behaviour of giving way to pedestrians (whose achivement is denoted by `givenWaypPed`). Such a behaviour is 'implemented' by the human as modelled by the following deliberate basic activity

```
goal(driving,givenWayPed) :
     none | none --> stop |> waitForFree duration d4
```

where `stop` is the action of stopping the car and `waitForFree` denotes the driver's mental state of waiting for the zebra crossing to be free.

Once automaticity in driving is acquired, the driver is no longer aware of low-level details and resorts to implicit attention to perform them (automatic control). In general, also an expert driver always starts driving with a precise goal in mind, which normally is that of reaching a specific destination, possibly as a subgoal of the reason for reaching it. Although such a goal is kept in the driver's STM, most driving activities are carried out under automatic control, with no need to retrieve the learned rules. Therefore, the behaviour of an expert driver is modelled as follows

```
automatism(driving) : none | zebra --> noAction |> zebra duration d1
automatism(driving) : none | ped --> noAction |> ped duration d2
automatism(driving) :
     moving ; zebra |> ped --> stop |> ped ; waitForFree duration d3
automatism(driving) :
     moving ; ped |> zebra --> stop |> zebra ; waitForFree duration d3
```

The first two automatic activities model implicit attention, which results in the storage of the perception of zebra crossing and pedestrians, respectively. The last two automatic activities model the automatic reaction to the perception of pedestrian in combination with the awareness of the presence of a zebra crossing or the perception of zebra crossing in combination with the awareness of the presence of pedestrian, depending on which perception occurs first.

We can note that automatic behaviour is more efficient than deliberate behviour for the following reasons:

- there are no goals in STM to drive explicit attention (low cognitive load);
- there is an immediate reaction to perceptions, when in the appropriate mental state (faster reaction);
- there is no recourse to inference (decreased access to LTM).

4 Dynamics of BRDL Models

We model the structure of the human memory using the following Real-time Maude class.

```
class Human | cognitiveLoad : Nat,
              shortTermMemory : TimedItemSet,
              inferSemMem : InferMem,
              activSemMem : ActivMem,
              procMem : ProcMem .
```

The STM is modelled by attribute `shortTermMemory` with `cognitiveLoad` being its current load, the semantic memory by the two attributes `inferSemMem` and `activSemMem` and the procedural memory by the single attribute `procMem` .

4.1 STM Model with Real-time Maude

The limited capacity of STM requires the presence of a mechanism to empty it when the stored information is no longer needed. In fact, information in STM decay very quickly, normally in less than one minute, unless it is reinforced through maintenance rehearsal. To implement STM decay, we need to associate a time to the elements of sort `Item`

```
sorts TimedItem TimedItemSet .
subsort TimedItem < TimedItemSet .
op _decay_ : Item Time -> TimedItem [ctor] .
op emptyTIS : -> TimedItemSet [ctor] .
op _;_ : TimedItemSet TimedItemSet -> TimedItemSet
         [ctor assoc comm id: emptyTIS] .
op maxDecayTime : -> Time .
eq maxDecayTime = 20000 .
```

Therefore STM is modelled as an element of sort `TimedItemSet`, set of elements of sort `TimedItem`. A piece of information in STM is associated with a *decay time*, which is initialised to the *maximum decay time* (`maxDecayTime`, for example set to 20 000 msec.) when the information is stored in STM. Then decay time decreases along with the passage of time. A piece of information disappears from the STM once its decay time has decreased to 0.

Additionally, every time a goal is achieved, there may be a subconscious removal of information from STM, a process called *closure*: the information used to complete the task is likely to be removed from the STM, since it is no longer needed. Therefore, when closure occurs, a piece of information may disappear from the STM even before its decay time has decrease to 0. Conversely, maintenance rehearsal reset the decay time to the value of the maximum decay time.

In order for a goal with `BIS` as parameter of sort `BasicItemSet` to be achieved

- the entire cognitive information included in `BIS` has to be in STM;
- one of the perceptions (if any) has to be the trigger of the occurring basic activity (which may be automatic or deliberate);

– one of the actions (if any) has to be performed by the occurring basic activity.

This is implemented by operations

```
op removeTime : TimedItemSet -> ItemSet .
op goalAchieved : Goal ItemSet DefPerception DefAction -> Bool .
```

where operation `removeTime` removes the time from the elements of the STM and operation `goalAchieved` returns `true` if the goal is achieved.

It is not fully understood how closure works. We can definitely say that once the goal is achieved, it is removed from the STM. However, it is not clear what happens to the information that was stored in STM in order to achieve the goal. We said at the end of Section 3.1 that if an ATM is used with the goal of withdrawing cash, the goal is achieved when the user collects the cash delivered by the ATM [6]. However, old ATM interfaces (some still in activity) deliver the cash before returning the card to the user. There is then the possibility that the user collects the cash and, feeling the goal achieved, abandons the interaction forgetting to collect the card. This cognitive error is known as *post-completion error* [4, 5, 11]. It could be explained by the loss of the information that was stored in STM when the user inserted the card in the ATM, as a reminder to collect the card at a later stage. In fact, such a loss of information is the result of the closure due to the achievement of the goal when the user collects the cash.

In practice, however, a user interacting with an old ATM interface does not always forget the card. This may be explained by assuming that the likelihood to forget the card depends on the user's cognitive load. Therefore we define the following thresholds

```
op closureThresholdLow : -> Nat . eq closureThresholdLow = 4 .
op closureThresholdHigh : -> Nat . eq closureThresholdHigh = 6 .
```

and force closure to occurs if the cognitive load is at least `closureThresholdHigh` and prevent its occurrence if the cognitive load is less than `closureThresholdLow`. In all other cases closure may occur non-deterministically.

Finally, a piece of information may also non-deterministically disappear from the STM when the STM has reached its maximum capacity and it is needed to make space for the storage of new information. This is implemented by the following rewrite rule

```
crl [forgetSomethingIfSTMfull] :
      < H : Human | shortTermMemory : (ITEM decay NZT) ; STM,
                    cognitiveLoad : CL >
    =>
       < H : Human |  shortTermMemory : STM,
                      cognitiveLoad : sd(CL, 1) >
     if CL > stmCapacity .
```

where `sd` is the symmetric difference between natural numbers.

4.2 Model of the Environment

A specific environment with which the human interacts is defined as an object of class

```
  class Environment | state : TimedEnvState,
                      transitions : EnvTransitions .
```

The `state` attribute characterises the environment and its time aspects by means of the following sort structure

```
sort EnvState .
sorts TimedEnvState ExpiringEnvState TimedEnvStateSet .
subsort EnvState < ExpiringEnvState < TimedEnvState < TimedEnvStateSet .
op _expiring‘in_ : EnvState Time -> ExpiringEnvState [ctor] .
op _expired : EnvState -> ExpiringEnvState [ctor] .
op _in‘time_ : ExpiringEnvState Time -> TimedEnvState [ctor right id: 0]
var STATE : EnvState .
eq STATE expiring in 0 = STATE expired .
op noEnvState : -> TimedEnvStateSet [ctor] .
op _;_ : TimedEnvStateSet TimedEnvStateSet -> TimedEnvStateSet
                  [ctor assoc comm id: noEnvState] .
```

where

- sort `EnvState` of environmental states is user-defined and application-specific;
- sort `ExpiringEnvState` add a *life time* to the environmental state;
- sort `TimedEnvState` add a *delay time* to the (possibly expiring) environmental state.

Note that `0` is right identity in the construction of timed environmental states out of expiring environmental states. Thus a timed environmental state with delay `0` is actually an expiring environmental state (with no delay). Moreover, expiring environmental states are characterised by a postfix constructor `expired` in order to determine different transitions with respect to the non-expired states.

Sort `EnvTransitions` models environmental transitions as follows.

```
 sort EnvTransitions .
 sort EnvTransition .
 subsort EnvTransition < EnvTransitions .
 op noTrans : -> EnvTransitions [ctor] .
 op _-->_ : ExpiringEnvState TimedEnvState -> EnvTransition [ctor] .
 op _--_-->_ : EnvState Action TimedEnvState -> EnvTransition [ctor] .
 op _;_ : EnvTransitions EnvTransitions -> EnvTransitions
                  [ctor assoc comm id: noTrans] .
```

Obviously interactions (`_--_-->_`) are associated with actions, internal actions (`_-->_`) are not.

Sort `EnvTransitions` is populated through user-defined, application-specific operation

```
op transitions : Cid Oid -> EnvTransitions .
```

where `Cid` is a class identifier and `Oid` is an object identifier.

States of the environment may be observable by humans. Such observability is modelled as

```
op observability : ExpiringEnvState -> PerceptionSet .
eq observability(STATE expired) = none .
eq observability(STATE expiring in NZT) = observability(STATE) .
```

with the rest of operation `observability` user-defined and application-specific.

4.3 'Zebra Crossing' Environment

In order to define the behaviour of the environment for the example in Section 3.3, we need two environments, one to model the car behaviour and one to model the zebra crossing behaviour. Both car and zebra crossing have a location, which is variable for the car and fixed for the zebra crossing. They also need to have additional state components to characterise whether the car is moving or is static and whether the zebra crossing has pedestrians or is free.

Environment and Observability If we assume to have only one human, one car and one zebra crossing

```
ops driver1 car1 zebra1 : -> Oid [ctor] .
```

then the environmental state is defined as follows.

```
sorts Location AdditionalState .
ops atInit atZebra atFinal : -> Location [ctor] .
ops hasPed isFree isMoving isBraking isStatic : -> AdditionalState [ctor] .
op state : Location AdditionalState -> EnvState [ctor] .
```

The meanings of the operations that define locations and additional state components are obvious. An environmental state consists of a location and an additional state.

The observability operation is defined as follows

```
eq observability(state(LOC,isStatic)) = static .
eq observability(state(LOC,isMoving)) = moving .
eq observability(state(zebra1,AS)) = zebra .
eq observability(state(zebra1,isFree)) = zebra ;  noPed .
eq observability(state(zebra1,hasPed)) = zebra1 ; ped .
```

Transition System The environmental transition systems are defined as

```
class Car . subclass Car < Environment .
var C : Oid .
eq transitions(Car, C) =
   (state(atInit,isMoving) --> state(atZebra, isMoving)
                                    expiring in 1 in time 30000) ;
   (state(atZebra,isMoving) -- stop(C) --> state(atZebra, isBraking)) ;
   (state(atZebra,isBraking) --> state(atZebra, isStatic) in time 2000) ;
   (state(atZebra,isStatic) -- go(C) --> state(atZebra, isMoving)) ;
   (state(atZebra,isMoving) --> state(atFinal, isMoving)
                                    expiring in 1 in time 30000) ;
   (state(atFinal,isMoving) -- stop(C) --> state(atFinal, isBraking)) ;
   (state(atFinal,isBraking) --> state(atFinal, isStatic in time 2000) .
```

for the car, and

```
class Zebra . subclass Zebra < Environment .
var Z : Oid .
eq transitions(Zebra, Z) =
   (state(Z,isFree) --> state(Z, hasPed) expiring in 5000) ;
   (state(Z,hasPed) --> state(Z, isFree) expiring in  20000) .
```

for the zebra crossing.

The timings mean that the car takes time 30000 to move between two consecutive locations and time 2000 to brake, being in an unstable state until these times are elapsed and, once stable, expiring immediately (in time 1) if not taken, and that there are pedestrian crossing every 25000 time units ($25000 = 20000 + 5000$) who take time 5000 to cross.

Initial Configuration Let us consider a driver who has already acquired a general automatism in driving, in which implicit attention controls the storage of information in STM, but still needs to perform inferences to apply road code rules. The initial configuration of the overall system is

```
op init : -> Configuration .
eq init = < cerone : Human |
   cognitiveLoad : 2,
   shortTermMemory : emptyTIS,
   proceduralMemory :
(automatism(driving) : none | moving --> noAction |> moving duration 1) ;
(automatism(driving) : none | static --> noAction |> static duration 1) ;
(automatism(driving) : none | zebra --> noAction |> zebra duration 1) ;
(automatism(driving) : none | ped --> noAction |> hasPed duration 1) ;
(automatism(driving) : none | freePed --> noAction |> freePed duration 1),
   knowledge :
(infer(driving) : (moving ; zebra ; hasPed) |-->|>
     goal(driving,givenWayPed) duration 10) ;
(infer(driving) : (static ; zebra ; freePed) |-->|>
     goal(driving,leftZebraCrossing) duration 10),
   activity :
(goal(driving,givenWayPed) :
     (moving ; zebra ; hasPed) | noPerception -->
          stop(car1) |> waitForPed) duration 10) ;
     (goal(driving,leftZebraCrossing) :
          (zebra ; waitForPed) > (static ; freePed) | noPerception -->
               go(car1) |> none duration 10)
>
```

```
< zebra1 : Zebra | transitions : transitions(Zebra, zebra1),
                   state : state(zebra1,zebraPed) expiring in 5000
>
< car1 : Car | transitions : transitions(Car, car1),
               state : state(initLoc,moving)
> .
```

5 Rewrite Rules

At the end of Section 4.1 we have introduced the `forgetSomethingIfSTMfull` rewrite rule. In this section we illustrate three more rewrite rules: `internal`, `reasoning` and `timePassing`. Other rewrite rules not presented here involve the automatic and deliberated activities, including special cases such as implicit and explicit attention, when the action is absent, and cognition, when both perception and action are absent, which are duplicated for the closure and non-closure cases.

5.1 Internal Action Rewrite Rule

Internal actions are modelled by the following rewrite rule.

```
crl [internal] :
      {< E : Environment | >
      REST}
    =>
      {< E : Environment | state : TESTATE >
      REST}
   if ALL-TESTATES := fireTransitions(< E : Environment | >)
      /\ TESTATE ; OTHER-TESTATES := ALL-TESTATES .
```

The rule makes use of the `fireTransitions` operation, which is defined as follows

```
op fireTransitions : Configuration -> TimedEnvStateSet .
eq fireTransitions(< E : Environment |
             state : ESTATE,
             transitions : (ESTATE  --> TESTATE) ; TRANSES > REST ) =
  TESTATE ; fireTransitions( < E : Environment |
             state : ESTATE,
             transitions : TRANSES > REST ) .
eq fireTransitions( REST ) = noEnvState [owise] .
```

and returns the set of the environmental states generated by the firing of the enabled internal transitions. In the `internal` rewrite rule, such a set is assigned to variable `ALL-TESTATES`, which is matched to `TESTATE ; OTHER-TESTATES`, thus giving the rewritten state `TESTATE`.

5.2 Reasoning Rewrite Rule

Reasoning is modelled by the following rewrite rule.

```
crl [reasoning] :
      {< H : Human |
         cognitiveLoad : CL,
         shortTermMemory : (TIS1 ; TIS2),
         inferMem : (infer(KD) : BIS >| -->|> IS duration T) ; KNOW >
       REST}
    =>
      {< H : Human |
         cognitiveLoad : card(NEW-STM),
         shortTermMemory : NEW-STM,
         inferMem : (infer(KD) : BIS >| -->|> IS duration T) ; KNOW >
       idle(REST, T)}
   in time T
   if BIS == removeTime(TIS1)
    /\ CL < closureThresholdHigh /\ CL <= stmCapacity
    /\ NEW-STM := addTime(BIS ; IS, maxDecayTime)) ; idle(TIS2,T) .
```

In addition to operation `removeTime` introduced in Section 4.1, the rule makes use of

– the `addTime` operation, which transforms the untimed sets `BIS` and `IS` into a timed set to be added to the STM;
– the `idle` operation, which models the passage of a given time by decrementing each element of sort `TimedItemSet` of the STM and, for each environment component, the delay and expiration times of the `state` attribute, which is of sort `TimedEnvState`, if positive.

Note that also the decay time of the premises in `BIS` is set to the maximum decay time because the use of `BIS` in the inference is an implicit maintenance rehearsal of its timed version `TIS1`.

Let us consider the zebra crossing example introduced in Sections 3.3 and 4.3. When `moving`, `zebra` and `ped` are stored in the STM, the road code rule concerning zebra crossing (from Section 4.3)

```
inference(driving) :
     moving ; zebra ; ped |-->|> goal(driving,givenWayPed) duration d3
```

which enables the application of Maude `reasoning` conditional rule with

```
BIS = moving ; zebra ; ped     and     IS = goal(driving,givenWayPed)
```

Thus the new goal `goal(driving,givenWayPed)` is added to the STM and triggers the following deliberate basic activity, stored in LTM, which implement the road code rule (from Section 4.3)

```
goal(driving,givenWayPed) :
     none | none --> stop |> waitForFree duration d4
```

which dictates the action of stopping the car (`stop`) and the storage of `waitForFree` in the STM.

5.3 Time Passing Rewrite Rule

```
crl [timePassing] :
      {CONFIG}
    =>
      {idle(CONFIG,1)}
      in time 1
   if nothingEnabled(CONFIG) .
```

where operation `nothingEnabled` is defined as

```
op nothingEnabled enablingSTM : Configuration -> Bool .
eq nothingEnabled(CONFIG) = (fireTransitions(CONFIG) == noEnvState)
                            and (enablingSTM(CONFIG)) == false .
```

and operation `enablingSTM` checks whether the configuration has an object of class `Human` whose STM either exceeds the maximum cognitive load or is enabling an inference rule, an automatic basic activity or a deliberate basic activity. In this way the `timePassing` rewrite rule may be applied only if no other rewrite rule can be applied.

6 Conclusion and Future Work

We have presented a translation of BRDL into Real-time Maude. In previous work [6, 7], a subset of BRDL, the *Human Behaviour Description Language (HBDL)*, was implemented using Core Maude. However, that untimed implementation was limited to automatic and deliberate behaviour powered by a very simple, fixed short-term memory model, with a minimalist inflexible approach to closure and without decay. Reasoning and problem solving aspects had to be modelled explicitly in a procedural way in a limited, unstructured environment consisting of just one component.

BRDL, instead, is equipped with the linguistic constructs to specify reasoning goals, inference rules and unsolved problems. The Real-time Maude implementation of BRDL presented in this paper provides an engine capable to emulate the human reasoning specified by such constructs. Moreover, the object-oriented and real-time aspects of Maude allow us to overcome the limitation of previous work [6] and carry out the implementation of the time aspects envisaged in recent work [7].

As future work we plan to implement BRDL problem solving constructs [8] and use the model checking capabilities of Real-time Maude to extend the untimed analysis approach used in previous work [6] to the formal verification of timed properties.

References

1. Atkinson, R.C., Shiffrin, R.M.: Human memory: A proposed system and its control processes. In: Spense, K.W. (ed.) The psychology of learning and motivation: Advances in research and theory II, pp. 89–195. Academic Press (1968)

2. Atkinson, R.C., Shiffrin, R.M.: The control of short-term memory. Scientific American **225**(2), 82–90 (1971)
3. Butterworth, R., Blandford, A.E., Duke, D.: Demonstrating the cognitive plausability of interactive systems. Form. Asp. of Comput. **12**, 237–259 (2000)
4. Byrne, M.D., Bovair, S.: A working memory model of a common procedural error. Cognitive Science **21**, 31–61 (1997)
5. Byrne, M.D., Davis, E.M.: Task structure and postcompletion error in the execution of a routine procedure. Human Factors **48**, 627–638 (2006)
6. Cerone, A.: A cognitive framework based on rewriting logic for the analysis of interactive systems. In: Software Engineering and Formal Methods (SEFM 2016), pp. 287–303. No. 9763 in Lecture Notes in Computer Science, Springer (2016)
7. Cerone, A.: Towards a cognitive architecture for the formal analysis of human behaviour and learning. In: STAF collocated Workshops 2018 (FMIS), pp. 216–232. No. 11176 in Lecture Notes in Computer Science, Springer (2018)
8. Cerone, A.: Behaviour and reasoning description language (BRDL). In: SEFM 2019 Collocated Workshops. In press. Lecture Notes in Computer Science, Springer (2019)
9. Harrison, M.D., Campos, J.C., Rukšėnas, R., Curzon, P.: Modelling information resources and their salience in medical device design. In: EICS 2016, pp. 194–203. ACM (2026)
10. Kotseruba, I., Tsotsos, J.K.: 40 years of cognitive architectures: core cognitive abilities and practical applications. Artificial Intelligence Review (2018), https://doi.org/10.1007/s10462-018-9646-y
11. Li, S.W., Blandford, A., Cairns, P., Young, R.M.: The effect of interruptions on postcompletion and other procedural errors: An account based on the activation-based goal memory model. Journal Of Experimental Psychology: Applied **14**, 314–328 (2008)
12. Martí-Oliet, N., Meseguer, J.: Rewriting logic: roadmap and bibliography. Theoretical Computer Science **285**(2), 121–154 (2002)
13. Norman, D.A., Shallice, T.: Attention to action: Willed and automatic control of behaviour. In: Consciousness and Self-Regulation, Advances in Research and Theory, vol. 4. Plenum Press (1986)
14. Ölveczky, P.C.: Real-time Maude and its applications. In: Proc. of WRLA 2014, Lecture Notes in Computer Science, vol. 8663, pp. 42–79. Springer (2014)
15. Ölveczky, P.C.: Designing Reliable Distributed Systems. Undergraduate Topics in Computer Science, Springer (2017)
16. Ölveczky, P.C., Meseguer, J.: Real-time Maude 2.1. In: Rewriting Logic and Its Applications 2004 (WRLA 2004), vol. 117, pp. 285–314. Elsevier (2005)
17. Oulasvirta, A., Kristensson, P., Bi, X., Howes, A. (eds.): Computational Interaction. Oxford University Press (2018)
18. Weyers, B., Bowen, J., Dix, A., Palanque, P. (eds.): The Handbook of Formal Methods in Human-Computer Interaction. Springer (2017)

Author Index

Keyword Index