Electronic Communications of the EASST
Volume (FMIS09 Preliminary Proceedings)


# Preliminary Proceedings of the Third International Workshop on Formal Methods for Interactive Systems (FMIS 2009)

Guest Editors: Michael Harrison, Mieke Massink
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

# Preface

This third edition of the International Workshop on Formal Methods for Interactive Systems (FMIS 2009) is a forum for the presentation and discussion of research in the interface between formal methods and interactive system design. This subfield, within applied formal methods, provides interesting challenges to the specification and analysis of systems: what features of interactive systems can be specified that contribute to an understanding of their usability? what analytic techniques are appropriate to assessing whether the design represented by the specification has appropriate properties relating to its use? The theme of the workshop is admirably scoped by the keynote address: "Who wants a model and why?" in which Muffy Calder reflects on the role of models in recent analyses relating to pervasive and uniquitous systems.

The papers presented here combine traditional concerns of formal methods in HCI with novel concerns associated with more recent software applications. Hence Bowen and Reeves' paper focuses on the model based development problem and considers the role of testing in relation to the specification of interactive systems, while Combéfis is concerned with the integration of knowledge about user tasks and the operating environment into the system model. These are traditional concerns and it is encouraging to see continuing development of these themes. Several of the papers deal with emerging mainstream application areas including ubiquitous systems in general and techniques for their analysis (Arapinis and others, Calder and others), social networking (Catano and others) and enterprise systems (Overbeek and others). These applications require novel analysis from the perspective of human computer interaction. They require focus on interoperability (Arapinis and others), context awareness (Calder and others), multiple viewpoints (Overbeek and others) and privacy (Catano and others). Further papers in the collection focus on particular frameworks and analytic techniques that are required to model and analyse trust-related emotion (Bonnefon and others) and the stochastic properties of software based systems (Anderson and others).

Previous workshops in this series were held in Macau (October 2006) and Lancaster (September 2007). This year FMIS is co-located with the 16th International Symposium on Formal Methods. It is a privilege to be co-located with this international forum for researchers, practitioners and educators in the field of formal methods which is held in Eindhoven, the Netherlands.

We would like to thank all the members of the Programme Committee and the additional referees for their careful and timely evaluation and discussion of the submitted papers. We are grateful to the FM2009 Conference for hosting the FMIS 2009 this year and taking care of many organisational aspects, and to FM Europe for its financial support. We would also like to thank all authors for their submissions without which this workshop could not have taken place. Additionally, we thank EASST (European Association of Software Science Technology), and our home institutions Newcastle University and CNR-ISTI for their support.

We hope that you will find this programme interesting and thought-provoking and that the workshop will provide you with a valuable opportunity to share ideas with other researchers and practitioners from institutions around the world.

October, 2009

Michael Harrison
Newcastle University
FMIS09 Co-Chair

Mieke Massink
CNR-ISTI
FMIS09 Co-Chair

# Organisation

### Programme Committee Chairs

| | |
|---|---|
| Michael Harrison | Newcastle University, UK |
| Mieke Massink | National Research Council, CNR-ISTI, Italy |

### Programme Committee

| | |
|---|---|
| Ann Blandford | UCL Interaction Center, UK |
| Judy Bowen | University of Waikato, New Zealand |
| Paul Cairns | University of York, UK |
| José Creissac Campos | University of Minho, Portugal |
| Antonio Cerone | UNI-IIST, Macau SAR China |
| Paul Curzon | Queen Mary, University of London, UK |
| Alan Dix | Lancaster University, UK |
| Gavin Doherty | Trinity College, University of Dublin, Ireland |
| David Duce | Oxford Brookes University, Oxford, UK |
| Stefania Gnesi | CNR-ISTI, Pisa, Italy |
| Michael Harrison | Newcastle University, UK |
| C. Michael Holloway | NASA Langley Research Center, USA |
| Chris Johnson | University of Glasgow, UK |
| Mieke Massink | CNR-ISTI, Pisa, Italy |
| Philippe Palanque | University of Toulouse III, France |
| Luca Simoncini | University of Pisa, Italy |
| Daniel Sinnig | Concordia University, Canada |
| Harold Thimbleby | University of Wales Swansea, Wales |

### External Referees

| | |
|---|---|
| Alessandro Fantechi | University of Florence, Italy |

# Contents

**Regular papers:**



**Short papers:**

# A Logical Framework for Trust-Related Emotions

**Jean-François Bonnefon[1], Dominique Longin[2] and Manh-Hung Nguyen[3]**

[1] bonnefon@univ-tlse2.fr
University of Toulouse, CNRS, CLLE, France

[2] Dominique.Longin@irit.fr
University of Toulouse, CNRS, IRIT, France

[3] Manh-Hung.Nguyen@irit.fr
University of Toulouse, UPS, IRIT, France

**Abstract:** Emotion and trust are two important concerns for the elaboration of interaction systems that would be closer and more attractive to their users, in particular by endowing machines with the ability to predict, understand, and process emotions and trust. This paper attempts to construct a common logical framework for the representation of emotion and trust. This logical framework combines a logic of belief and choice, a logic of time, and a dynamic logic. Using this common framework, we identify formal relations between trust and emotions, for which we also provide behavioral validation.

**Keywords:** Modal logic, emotions, trust, distrust

## 1 Introduction

The rapidly growing field of affective computing aims at developing interaction systems that are closer and more attractive to their users, in particular by endowing machines with the ability to predict, understand, and process emotions (on the one hand), and trust (on the other hand). In this article, we introduce a unified logical approach to represent the cognitive structure of some emotions, of trust/distrust, and their relations at a formal level.

We formalize the concepts of emotions as well as trust/distrust based on cognitive models proposed by cognitive psychologists. Regarding emotions, we draw on cognitive theories (for more detail, see [SSJ01]) which assume that emotions are closely tied to changes in beliefs and desires. We capitalize on psychological models that allow to recognize and distinguish emotions based on their decomposition in cognitive factors particularly the cognitive structure of emotion of Ortony et al. [OCC88], the cognitive patterns of emotion of Lazarus [Laz91] and the belief-desire theory of emotion (BDTE) [Rei09, Dre95]. Similarly, we attempt to adhere closely to cognitive definition of trust [CF01] and distrust [CFL08].

Although there are tight conceptual connections between emotion and trust [Lah01], and although trust [HLH+08] and emotions [AHL09] have been separately formalized, there is not yet a common logic to represent them both. Our work aims at filling that gap by formally representing trust and emotions in a common logic; this common logic will enable us to lay bare the formal relations between trust and emotion. The logic we offer is a combination of the logic of

beliefs and choices [HL04] (a refinement of [CL90]), the logic of time (introduced by Arthur Prior [Pri57]), and dynamic logic [FL79, HKT00].

This paper is organized as follows: Part 2 introduces the logical framework. Part 3 formalizes some emotions, Part 4 formalizes trust and distrust. Part 5 shows some formal relations between emotions and trust, and provides behavioral validation for these relations.

## 2 Logical Framework

**Syntax.** The syntactic primitives of our logic are as follows: a nonempty finite set of agents $AGT = \{i_1, i_2, \ldots, i_n\}$, a nonempty finite set of atomic events $EVT = \{e_1, e_2, \ldots, e_p\}$, and a nonempty set of atomic propositions $ATM = \{p_1, p_2, \ldots\}$. The variables $i$, $j$, $k$... denote agents. The expression $i_1{:}e_1 \in AGT \times EVT$ denotes an event $e_1$ intentionally caused by agent $i_1$ and $e_1$ is thus called an "action". The variables $\alpha$, $\beta$... denote such actions. The language of our logic is defined by the following BNF :

$$\varphi ::= p \mid i{:}\alpha\text{-}happens \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathtt{X}\varphi \mid \mathtt{X}^{-1}\varphi \mid \mathtt{G}\varphi \mid \mathtt{Bel}_i\,\varphi \mid \mathtt{Choice}_i\,\varphi \mid \mathtt{Grd}_I\varphi$$

where $p$ ranges over $ATM$, $i{:}\alpha$ ranges over $AGT \times EVT$, $i{:}\alpha\text{-}happens$ ranges over $ATM$, and $I \subseteq AGT$. The classical boolean connectives $\wedge$ (conjunction), $\rightarrow$ (material implication), $\leftrightarrow$ (material equivalence), $\top$ (tautology) and $\bot$ (contradiction) are defined from $\neg$ (negation) and $\vee$ (disjunction).

$i{:}\alpha\text{-}happens$ reads "agent $i$ is just about to perform the action $\alpha$"; $\mathtt{X}\varphi$ reads "$\varphi$ will be true next time"; $\mathtt{X}^{-1}\varphi$ reads "$\varphi$ was true at the previous time"; $\mathtt{G}\varphi$ reads "henceforth, $\varphi$ is true"; $\mathtt{Bel}_i\,\varphi$ reads "agent $i$ believes that $\varphi$ is true"; $\mathtt{Choice}_i\,\varphi$ reads "agent $i$ prefers that $\varphi$ be true"; $\mathtt{Grd}_I\varphi$ reads "$\varphi$ is publicly grounded between the agents in group I" (It is nothing else that a standard common belief operator). We define the following abbreviations:

$$i{:}\alpha\text{-}done \stackrel{def}{=} \mathtt{X}^{-1} i{:}\alpha\text{-}happens \qquad (\mathrm{Def}_{i:\alpha\text{-}done})$$

$$\mathtt{Happens}_{i:\alpha}\varphi \stackrel{def}{=} i{:}\alpha\text{-}happens \wedge \mathtt{X}\varphi \qquad (\mathrm{Def}_{\mathtt{Happens}_{i:\alpha}})$$

$$\mathtt{After}_{i:\alpha}\varphi \stackrel{def}{=} i{:}\alpha\text{-}happens \rightarrow \mathtt{X}\varphi \qquad (\mathrm{Def}_{\mathtt{After}_{i:\alpha}})$$

$$\mathtt{Done}_{i:\alpha}\varphi \stackrel{def}{=} i{:}\alpha\text{-}done \wedge \mathtt{X}^{-1}\varphi \qquad (\mathrm{Def}_{\mathtt{Done}_{i:\alpha}})$$

$$\mathtt{F}\varphi \stackrel{def}{=} \neg\mathtt{G}\neg\varphi \qquad (\mathrm{Def}_{\mathtt{F}})$$

$$\mathtt{Goal}_i\,\varphi \stackrel{def}{=} \mathtt{Choice}_i\,\mathtt{FBel}_i\,\varphi \qquad (\mathrm{Def}_{\mathtt{Goal}_i})$$

$$\mathtt{Intend}_i\,(i{:}\alpha) \stackrel{def}{=} \mathtt{Choice}_i\,\mathtt{F}i{:}\alpha\text{-}happens \qquad (\mathrm{Def}_{\mathtt{Intend}_i})$$

$$\mathtt{Capable}_i(i{:}\alpha) \stackrel{def}{=} \neg\mathtt{After}_{i:\alpha}\bot \qquad (\mathrm{Def}_{\mathtt{Capable}_i})$$

$$\mathtt{Possible}_i\varphi \stackrel{def}{=} \neg\mathtt{Bel}_i\neg\varphi \qquad (\mathrm{Def}_{\mathtt{Possible}_i})$$

$$\mathtt{Awareness}_i\varphi \stackrel{def}{=} \mathtt{X}^{-1}\neg\mathtt{Bel}_i\,\varphi \wedge \mathtt{Bel}_i\,\varphi \qquad (\mathrm{Def}_{\mathtt{Awareness}_i})$$

$i{:}\alpha\text{-}done$ reads "agent $i$ has done action $\alpha$"; $\mathtt{Happens}_{i:\alpha}\varphi$ reads "agent $i$ is doing action $\alpha$ and $\varphi$ will be true next time"; $\mathtt{After}_{i:\alpha}\varphi$ reads "$\varphi$ is true after any execution of $\alpha$ by $i$"; $\mathtt{Done}_{i:\alpha}\varphi$

reads "agent $i$ has done action $\alpha$ and $\varphi$ was true at previous time"; $\mathtt{F}\varphi$ reads "$\varphi$ will be true in some future instants"; $\mathtt{Goal}_i\,\varphi$ reads "agent $i$ has the goal (chosen preference) that $\varphi$ be true"; $\mathtt{Intend}_i\,(i{:}\alpha)$ reads "agent $i$ intends to do $\alpha$"; $\mathtt{Capable}_i(i{:}\alpha)$ reads "agent $i$ is capable to do $\alpha$"; $\mathtt{Possible}_i\varphi$ reads "agent $i$ believes that it is possible $\varphi$"; $\mathtt{Awareness}_i\varphi$ reads "agent $i$ has just experienced that $\varphi$ is true".

**Semantics.** We use a semantics based on linear time described by a history of time points. (This semantics is very closed to CTL* [CES86]) A frame $\mathscr{F}$ is a 4-tuples $\langle H,\mathscr{B},\mathscr{C},\mathscr{G}\rangle$ where: $H$ is a set of histories that are represented as sequences of time points, where each time point is identified by an integer $z\in\mathbb{Z}$, a time point $z$ in a history $h$ is called a situation $<h,z>$; $\mathscr{B}_i(h,z)$ denotes the set of histories believed as being possible by the agent $i$ in the situation $<h,z>$; $\mathscr{C}_i(h,z)$ denotes the set of histories chosen by the agent $i$ in the situation $<h,z>$; $\mathscr{G}_I(h,z)$ denotes the set of histories which are publicly grounded in the group I of agents, in the situation $<h,z>$.

All the accessibility relations $\mathscr{B}_i$ are serial, transitive and euclidean. All the accessibility $\mathscr{C}_i$ are serial. Moreover, we impose for every $z\in\mathbb{Z}$ that: if $h'\in\mathscr{B}_i(h,z)$ then $\mathscr{C}_i(h,z)=\mathscr{C}_i(h',z)$.

A model $\mathscr{M}$ is a couple $\langle\mathscr{F},\mathscr{V}\rangle$ where $\mathscr{F}$ is a frame and $\mathscr{V}$ is a function associating each atomic proposition $p$ with the set $\mathscr{V}(p)$ of couple $(h,z)$ where $p$ is true. Truth conditions are defined as follows:

$$\mathscr{M},h,z\models p \text{ iff } (h,z)\in\mathscr{V}(p)$$
$$\mathscr{M},h,z\models \mathtt{X}\varphi \text{ iff } \mathscr{M},h,z+1\models\varphi$$
$$\mathscr{M},h,z\models \mathtt{X}^{-1}\varphi \text{ iff } \mathscr{M},h,z-1\models\varphi$$
$$\mathscr{M},h,z\models \mathtt{G}\varphi \text{ iff } \mathscr{M},h,z'\models\varphi \text{ for every } z'\geq z$$
$$\mathscr{M},h,z\models \mathtt{Bel}_i\,\varphi \text{ iff } \mathscr{M},h',z\models\varphi \text{ for every } (h',z)\in\mathscr{B}_i(h,z)$$
$$\mathscr{M},h,z\models \mathtt{Choice}_i\,\varphi \text{ iff } \mathscr{M},h',z\models\varphi \text{ for every } (h',z)\in\mathscr{C}_i(h,z)$$
$$\mathscr{M},h,z\models \mathtt{Grd}_I\varphi \text{ iff } \mathscr{M},h',z\models\varphi \text{ for every } (h',z)\in\mathscr{G}_I(h,z) \text{ so that } \mathscr{G}_I=(\bigcup_{i\in I}\mathscr{B}_i)^+$$

$\mathscr{G}_I$ is the transitive closure of the belief accessibility relations. Other truth conditions are defined as usual.

**Axiomatics.** Due to our linear time semantics, the temporal operators satisfy the following principles:

$$i{:}\alpha\text{-}happens \leftrightarrow \mathtt{X}i{:}\alpha\text{-}done \tag{1}$$
$$\mathtt{X}\varphi\leftrightarrow\neg\mathtt{X}\neg\varphi \tag{2}$$
$$\varphi\leftrightarrow\mathtt{X}\mathtt{X}^{-1}\varphi \tag{3}$$
$$\varphi\leftrightarrow\mathtt{X}^{-1}\mathtt{X}\varphi \tag{4}$$
$$\mathtt{G}\varphi\leftrightarrow\varphi\wedge\mathtt{X}\mathtt{G}\varphi \tag{5}$$
$$\mathtt{G}(\varphi\rightarrow\mathtt{X}\varphi)\rightarrow(\varphi\rightarrow\mathtt{G}\varphi) \tag{6}$$

$\texttt{Bel}_i$ and $\texttt{Choice}_i$ operators are defined in a normal modal logic plus (D) axioms. Thus, if $\Box$ represents a $\texttt{Bel}_i$ operator or $\texttt{Choice}_i$ operator:

$$\frac{\varphi}{\Box\varphi} \qquad (\mathrm{RN}_\Box)$$

$$\Box(\varphi \rightarrow \psi) \rightarrow (\Box\varphi \rightarrow \Box\psi) \qquad (\mathrm{K}_\Box)$$

$$\Box\varphi \rightarrow \neg\Box\neg\varphi \qquad (\mathrm{D}_\Box)$$

($\mathrm{RN}_\Box$) means that all theorems are believed (respectively: chosen) by every agent $i$; ($\mathrm{K}_\Box$) means that beliefs (respectively: choices) are closed under material implication for every agent $i$; ($\mathrm{D}_\Box$) means that beliefs (respectively: choices) of every agent $i$ are rational: they cannot be contradictory.

The $\texttt{Bel}_i$ operators satisfy the following principles of introspection:

$$\texttt{Bel}_i\,\varphi \leftrightarrow \texttt{Bel}_i\texttt{Bel}_i\,\varphi \qquad (4_{\texttt{Bel}_i})$$

$$\neg\texttt{Bel}_i\,\varphi \leftrightarrow \texttt{Bel}_i\neg\texttt{Bel}_i\,\varphi \qquad (5_{\texttt{Bel}_i})$$

that mean that agent $i$ is conscious of its beliefs and of its disbeliefs.

The following principle follows from the semantical constraint between belief accessibility relation and choice accessibility relation, and from axiom ($\mathrm{D}_\Box$) for $\texttt{Bel}_i$:

$$\texttt{Choice}_i\,\varphi \leftrightarrow \texttt{Bel}_i\,\texttt{Choice}_i\,\varphi \qquad (4_{BC})$$

$$\neg\texttt{Choice}_i\,\varphi \leftrightarrow \texttt{Bel}_i\,\neg\texttt{Choice}_i\,\varphi \qquad (5_{BC})$$

that means that agent $i$ is conscious of its choices and of its dischoices.

The sound and complete axiomatization of $\texttt{Grd}_I$ operator is defined two following axioms:

$$\texttt{Grd}_I\varphi \leftrightarrow (EB_I\varphi \wedge EB_I\texttt{Grd}_I\varphi) \qquad (\mathrm{FP})$$

$$(EB_I\varphi \wedge \texttt{Grd}_I(\varphi \rightarrow EB_I\varphi)) \rightarrow \texttt{Grd}_I\varphi \qquad (\mathrm{LFP})$$

where $EB_I\varphi \stackrel{def}{=} \bigwedge_{i\in I}\texttt{Bel}_i\,\varphi$.

(FP) is the fixpoint axiom and (LFP) is the leant fixpoint axiom. Such $\texttt{Grd}_I$ operator has every properties of operators defined in the normal modal logic KD. Moreover, the following theorems hold for every agent $i,j$ in $I \subseteq AGT$:

$$\texttt{Grd}_I\varphi \leftrightarrow \texttt{Bel}_i\,\texttt{Grd}_I\varphi \qquad (4_{BG})$$

$$\texttt{Bel}_i\neg\texttt{Grd}_I\varphi \rightarrow \neg\texttt{Grd}_I\varphi \qquad (5_{BG})$$

$$\texttt{Grd}_I\varphi \rightarrow \texttt{Bel}_i\,\varphi \wedge \texttt{Bel}_j\,\varphi \qquad (7)$$

$$\texttt{Grd}_I\varphi \rightarrow \texttt{Bel}_i\texttt{Bel}_j\,\varphi \wedge \texttt{Bel}_j\texttt{Bel}_i\,\varphi \qquad (8)$$

$$\texttt{Grd}_I\varphi \rightarrow \texttt{Bel}_i\texttt{Bel}_j\texttt{Bel}_i\,\varphi \wedge \texttt{Bel}_j\texttt{Bel}_i\texttt{Bel}_j\,\varphi \qquad (9)$$

Theorem ($4_{BG}$) means that each member in group conscious about their public grounding. Theorem ($5_{BG}$) means that if a member in group conscious that there is no public grounding, then there is no public grounding. Theorem from (7) to (9) mean that if $\varphi$ is publicly grounded

in a group, then each member of group conscious about $\varphi$ and conscious that other member also conscious about $\varphi$, etc.

Linear time semantics entail the following principles:

$$\mathtt{G}\varphi \rightarrow \mathtt{After}_{i:\alpha}\varphi \tag{10}$$

$$\mathtt{Happens}_{i:\alpha}\varphi \rightarrow \mathtt{After}_{j:\beta}\varphi \tag{11}$$

$$\mathtt{After}_{i:\alpha}\varphi \leftrightarrow \neg\mathtt{Happens}_{i:\alpha}\neg\varphi \tag{12}$$

## 3 Formalization of the cognitive structure of emotion

In this section, we present the formalization of emotions, based on their cognitive structure as proposed by Ortony et al. [OCC88], Frijda [Fri86] as well as those of Reisenzei [Rei09] and Scherer et al. [Sch01].

**Joy/Distress.** The cognitive structure of *Joy* consists of two main factors: (i) an event $\varphi$ is desirable for agent $i$, and (ii) agent $i$ believes that event $\varphi$ just happened. To formalize the first factor, we consider that agent $i$ desiring event $\varphi$ means that $i$ wants $\varphi$ to be the case. So we formalize desire as a goal (chosen preference). Therefore, the first factor is potentially formalized as $\mathtt{Goal}_i\,\varphi$, the second factor may be formalized as $\mathtt{Bel}_i\,\varphi$.

However, we assume that emotion is triggered at the moment when all its factors are fulfilled, and that its intensity then decreases with time [dS01, Fri86]. Accordingly, we include the time factor into most emotional formulas. Thus, the first factor of *Joy* in particular means that at the previous instant, agent $i$ desired $\varphi$, until experiencing that $\varphi$ was in fact true: $\mathtt{X}^{-1}\mathtt{Goal}_i\,\varphi$. The second factor means that agent $i$ has just experienced that $\varphi$ is true and did not previously know it: $\mathtt{Awareness}_i\varphi$.

Moreover, we consider that in order to be joyful, agent $i$ must keep in mind his desire in the previous instant. It means that until now, $i$ believes about his desire: $\mathtt{Bel}_i\,\mathtt{X}^{-1}\mathtt{Goal}_i\,\varphi$. Hereafter, we add this analysis for almost emotional formulas.

The same analysis applies to *Distress*, except that in the first factor of *Distress*, event $\varphi$ is undesirable for agent $i$, which we assume to mean that agent $i$ desired event $\neg\varphi$: $\mathtt{X}^{-1}\mathtt{Goal}_i\neg\varphi$. We accordingly formalize the concept of *Joy* and *Distress*:

**Definition 1** (Joy/Distress)

$$\mathtt{Joy}_i\,\varphi \stackrel{def}{=} \mathtt{Bel}_i\,\mathtt{X}^{-1}\mathtt{Goal}_i\,\varphi \wedge \mathtt{Awareness}_i\varphi$$

$$\mathtt{Distress}_i\,\varphi \stackrel{def}{=} \mathtt{Bel}_i\,\mathtt{X}^{-1}\mathtt{Goal}_i\,\neg\varphi \wedge \mathtt{Awareness}_i\varphi$$

To illustrate the definition of *Joy*, we can say that an individual is joyful when he has just realized that he won the lottery ($\mathtt{Awareness}_{man}$*(win lottery)*) with the trivial assumption that he had been desiring to win the lottery ($\mathtt{X}^{-1}\mathtt{Goal}_{man}$*(win lottery)*). In contrast, to illustrate the definition of *Distress*, we can say that an individual feels distress when she learns she has lost her job ($\mathtt{Awareness}_{woman}$*(lost job)*) assuming that she had the goal not to lose her job ($\mathtt{X}^{-1}\mathtt{Goal}_{woman}\,\neg$*(lost job)*).

**Hope/Fear.** The cognitive structure of *Hope* consists of two factors: (i) an event $\varphi$ is desirable for agent $i$, and (ii) agent $i$ believes that event $\varphi$ may happen in the future. To formalize the first factor, we consider that event $\varphi$ has not yet happened at the moment when $i$ hopes for it: $\texttt{Goal}_i\,\varphi$.

We interpret the second factor, as meaning that among all of possible future worlds, agent $i$ believes that there is at least one world in which $\varphi$ will be the case. In other terms, agent $i$ does not believe that $\varphi$ will be false in all of possible future worlds: $\texttt{Possible}_i\texttt{F}\varphi$. If $i$ believes that $\varphi$ can never be the case in all of possible future worlds, then $i$ has no ground for hope.

The same analysis applies to *Fear*, except that event $\varphi$ is now undesirable for agent $i$: $\texttt{Goal}_i\,\neg\varphi$. We accordingly formalize the concept of *Hope* and *Fear*:

**Definition 2** (Hope/Fear)

$$\texttt{Hope}_i\,\varphi \stackrel{def}{=} \texttt{Goal}_i\,\varphi \wedge \texttt{Possible}_i\texttt{F}\varphi$$

$$\texttt{Fear}_i\,\varphi \stackrel{def}{=} \texttt{Goal}_i\,\neg\varphi \wedge \texttt{Possible}_i\texttt{F}\varphi$$

For example, a debutante is hopeful about being asked to dance, for she thinks it is possible ($\texttt{Possible}_{girl}\texttt{F}$*(being asked to dance)*) and this is what she wants ($\texttt{Goal}_{girl}$*(being asked to dance)*). In contrast, an employee fears to be fired when he does not wish to be fired ($\texttt{Goal}_{employee}\,\neg(fired)$) but believes it is a possibility $\texttt{Possible}_{employee}\texttt{F}$*(to be fired)*).

**Satisfaction/Disappointment.** The cognitive structure of *Satisfaction* consists of three factors: (i) agent $i$ desires event $\varphi$, (ii) agent $i$ used to believe that event $\varphi$ might happen in the near future, and (iii) agent $i$ now believes that event $\varphi$ really just happened. The first two factors mean that, at the previous instant, $i$ desired $\varphi$ ($\texttt{X}^{-1}\texttt{Goal}_i\,\varphi$), and $i$ believed that $\varphi$ could be true in the future ($\texttt{X}^{-1}\texttt{Possible}_i\texttt{F}\varphi$) (cf. the analysis of the second factor of *Hope*). The last factor means that $i$ now believes that $\varphi$ is true, but did not know it the previous instant ($\texttt{Awareness}_i\varphi$).

The only difference in the case of *Disappointment* is that, in the previous instant, agent $i$ desired event $\neg\varphi$ instead of $\varphi$ ($\texttt{X}^{-1}\texttt{Goal}_i\,\neg\varphi$), and that $i$ believed that $\neg\varphi$ was possibly true in the future ($\texttt{X}^{-1}\texttt{Possible}_i\texttt{F}\neg\varphi$). We formalize *Satisfaction* and *Disappointment* as

**Definition 3** (Satisfaction/Disappointment)

$$\texttt{Satisfaction}_i\,\varphi \stackrel{def}{=} \texttt{Bel}_i\,\texttt{X}^{-1}(\texttt{Goal}_i\,\varphi \wedge \texttt{Possible}_i\texttt{F}\varphi) \wedge \texttt{Awareness}_i\varphi$$

$$\texttt{Disappointment}_i\,\varphi \stackrel{def}{=} \texttt{Bel}_i\,\texttt{X}^{-1}(\texttt{Goal}_i\,\neg\varphi \wedge \texttt{Possible}_i\texttt{F}\neg\varphi) \wedge \texttt{Awareness}_i\varphi$$

For example, when the debutante realizes that she is indeed asked to dance ($\texttt{Awareness}_{girl}$*(asked to dance)*) she is satisfied. Were she not to be asked to dance ($\texttt{Awareness}_{girl}$*(not asked to dance)*), she would feel disappointed.

We can point out the relations between *Satisfaction*, *Disappointment* and *Hope*:

$$\texttt{Satisfaction}_i\,\varphi \leftrightarrow \texttt{Bel}_i\,\texttt{X}^{-1}\texttt{Hope}_i\,\varphi \wedge \texttt{Awareness}_i\varphi \tag{13}$$

$$\texttt{Disappointment}_i\,\varphi \leftrightarrow \texttt{Bel}_i\,\texttt{X}^{-1}\texttt{Hope}_i\,\neg\varphi \wedge \texttt{Awareness}_i\varphi \tag{14}$$

The relation between *Satisfaction* and *Joy* can be formalized as Proposition 1: if we feel satisfaction about something, then we will also feel joy about it.

**Proposition 1** *(Satisfaction implies Joy)*

$$\texttt{Satisfaction}_i\,\varphi \rightarrow \texttt{Joy}_i\,\varphi$$

**Fear-confirmed/Relief.** The cognitive structure of *Fear-confirmed* consists of three factors: (i) an event $\varphi$ was undesirable for agent $i$, (ii) agent $i$ believed that event $\varphi$ might happen in the near future, and (iii) agent $i$ now believes that event $\varphi$ really just happened.

We use the same analysis as for *Satisfaction*, except that in the previous instant, $\neg\varphi$ was desirable for agent $i$ ($\texttt{X}^{-1}\texttt{Goal}_i\,\neg\varphi$).

The difference in the case of *Relief* is that, in the previous instant, agent $i$ desired event $\varphi$ ($\texttt{X}^{-1}\texttt{Goal}_i\,\varphi$), and $i$ believed that $\neg\varphi$ might be true in the near future ($\texttt{X}^{-1}\texttt{Possible}_i\texttt{F}\neg\varphi$). We formalize *Fear-confirmed* and *Relief* as:

**Definition 4** (Fear-confirmed/Relief)

$$\texttt{FearConfirmed}_i\,\varphi \stackrel{def}{=} \texttt{Bel}_i\,\texttt{X}^{-1}(\texttt{Goal}_i\,\neg\varphi \wedge \texttt{Possible}_i\texttt{F}\varphi) \wedge \texttt{Awareness}_i\varphi$$

$$\texttt{Relief}_i\,\varphi \stackrel{def}{=} \texttt{Bel}_i\,\texttt{X}^{-1}(\texttt{Goal}_i\,\varphi \wedge \texttt{Possible}_i\texttt{F}\neg\varphi) \wedge \texttt{Awareness}_i\varphi$$

For example, the employee's fear of being fired is confirmed when he learns that he is indeed about to be fired ($\texttt{Awareness}_{employee}(fired)$) which he had been afraid of ($\texttt{X}^{-1}(\texttt{Goal}_{employee}\,\neg(fired) \wedge \texttt{Possible}_{employee}\texttt{F}(fired))$). In contrast, were he to learn that he is not going to be fired ($\texttt{Awareness}_{employee}(not\ fired)$), he would feel relief.

We can also point out the relations between *Fear-confirmed*, *Relief* and *Fear*:

$$\texttt{FearConfirmed}_i\,\varphi \leftrightarrow \texttt{Bel}_i\,\texttt{X}^{-1}\texttt{Fear}_i\,\varphi \wedge \texttt{Awareness}_i\varphi \tag{15}$$

$$\texttt{Relief}_i\,\varphi \leftrightarrow \texttt{Bel}_i\,\texttt{X}^{-1}\texttt{Fear}_i\,\neg\varphi \wedge \texttt{Awareness}_i\varphi \tag{16}$$

The relation between *Fear-confirmed* and *Distress* is stated in Proposition 2: if our fears about something are confirmed, then we feel distressed.

**Proposition 2** *(Fear-confirmed implies Distress)*

$$\texttt{FearConfirmed}_i\,\varphi \rightarrow \texttt{Distress}_i\,\varphi$$

# 4 Formalization of Trust

We now present the formalization of trust and distrust based on the cognitive definition of Castelfranchi and colleagues [CF01, CFL08].

**Trust.** We formalize the concept of trust based on Castelfranchi and Falcone's definition [CF01] of trust in action which says that agent $i$ trusts agent $j$ to ensure $\varphi$ by performing action $\alpha$ if and only if agent $i$ desires to achieve $\varphi$ ($\texttt{Goal}_i\,\varphi$), and agent $i$ expects that: (i) $\varphi$ can be achieved by doing action $\alpha$ ($\texttt{Bel}_i\,\texttt{After}_{j:\alpha}\varphi$); (ii) agent $j$ is able to perform action $\alpha$ ($\texttt{Bel}_i\,\texttt{Capable}_{j:\alpha}$); and (iii) agent $j$ has the intention to do such an action ($\texttt{Bel}_i\,\texttt{Intend}_j\,(j{:}\alpha)$).

However, these three factors are only necessary conditions, but not sufficient ones. For example, imagine that a robber wants to steal something located on the second floor of a mansion. There is a nurse on the first floor. The robber desires that the nurse stays where she is, because it makes his robbery possible. He also believes that it is possible that the nurse will stay where she is, and that it is actually her intention. Thus, the three conditions are satisfied, but we are reluctant nonetheless to say that the robber trusts the nurse to stay where she is in order to allow for his stealing, because there is no agreement between the nurse (trustee) and the robber (trustor). So here we need to add another condition for trust: an agreement between trustor and trustee that the trustee will perform such an action ($\texttt{Grd}_I\texttt{F}(trustee:\alpha)$), where $I = \{trustor, trustee\}$. We accordingly formalize the concept of trust as:

**Definition 5** (Trust)

$$\texttt{Trust}(i,j,\alpha,\varphi) \stackrel{def}{=} \texttt{Goal}_i\,\varphi \wedge \texttt{Bel}_i\,\texttt{After}_{j:\alpha}\varphi \wedge \texttt{Bel}_i\,\texttt{Capable}_{j:\alpha} \wedge \texttt{Bel}_i\,\texttt{Intend}_j\,(j{:}\alpha) \wedge \texttt{Grd}_{\{i,j\}}\texttt{F}\,j{:}\alpha\text{-}happens$$

For example, a boss trusts his secretary to prepare a report in order to present it at a company meeting because the boss desires the report ($\texttt{Goal}_{boss}(report)$), and in his opinion, the report can be possibly ready after the secretary prepares it ($\texttt{Bel}_{boss}After_{secretary:prepare}(report)$), the secretary has the ability and intention to prepare the report ($\texttt{Bel}_{boss}\,\texttt{Capable}_{secretary:prepare} \wedge \texttt{Bel}_{boss}\,\texttt{Intend}_{secretary}\,(secretary:prepare)$)). It is clear that in the relation between the boss and his secretary, there is an agreement that the secretary will prepare the report in time ($\texttt{Grd}_{boss,secretary}\texttt{F}secretary:prepare\text{-}happens$).

**Distrust.** We also adopt the definition of distrust given by Castelfranchi et al. [CFL08] which says that agent $i$ distrusts agent $j$ to ensure $\varphi$ by performing action $\alpha$ if and only if agent $i$ desires to achieve $\varphi$ ($\texttt{Goal}_i\,\varphi$), and agent $i$ believes that at least one of these conditions is fulfilled: (i) agent $j$ is not in the capacity to do action $\alpha$: $\texttt{Bel}_i \neg \texttt{After}_{j:\alpha}\varphi$, or (ii) agent $j$ is able to do $\alpha$ but he has not intention to do $\alpha$: $\texttt{Possible}_i\texttt{After}_{j:\alpha}\varphi \wedge \texttt{Bel}_i \neg \texttt{Intend}_j\,(j{:}\alpha)$ . We accordingly formalize this concept as:

**Definition 6** (Distrust)

$$\texttt{DisTrust}(i,j,\alpha,\varphi) \stackrel{def}{=} \texttt{Goal}_i\,\varphi \wedge (\texttt{Bel}_i \neg \texttt{After}_{j:\alpha}\varphi \vee (\texttt{Possible}_i\texttt{After}_{j:\alpha}\varphi \wedge \texttt{Bel}_i \neg \texttt{Intend}_j\,(j{:}\alpha)))$$

For example, in spite of desiring the report ($\texttt{Goal}_{boss}(report)$), the boss does not trust a new employee to prepare it because he believes the new employee is unable to perform that task($\texttt{Bel}_{boss} \neg \texttt{After}_{employee:prepare}(report)$).

From this definition, we can decompose the concept of distrust based only on the ability of trustee:

**Definition 7** (Distrust based on ability)

$$\texttt{C-DisTrust}(i,j,\alpha,\varphi) \stackrel{def}{=} \texttt{Goal}_i\,\varphi \wedge \texttt{Bel}_i \neg \texttt{After}_{j:\alpha}\varphi$$

# 5 Trust-Related Emotions

## 5.1 Formal Relations

**Trust and Hope.** *Trust* and *Hope* have an important relation because they both feature a positive expectation [CF01]. When $i$ trusts $j$, $i$ has a positive expectation about $j$'s power and performance. *Hope* also implies some positive expectation. The greater the expectations, the deeper the trust; and, conversely, the deeper the disappointment when expectations are unrealized [Bry07]. We formalize the former relation as Proposition 3, the latter as Proposition 5.

**Proposition 3** *(Trust implies Hope)*

$$\texttt{Trust}(i,j,\alpha,\varphi) \rightarrow \texttt{Hope}_i\varphi$$

This means that when we trust someone about an action that will bring some results, we are hopeful that the results will be obtained. For example, in a commercial transaction, when the buyer trusts his seller to send him a product after payment ($\texttt{Trust}(buyer, seller, send, receipt)$), he will be hopeful that he will receive the product ($\texttt{Hope}_{buyer}$ *receive product*). This proposition will be proved by applying Lemma 1: if we believe that $\varphi$ is true after every execution of action $\alpha$, and that someone is able to do $\alpha$, then we believe that there is at least a future world in which $\varphi$ is true.

**Lemma 1**

$$\texttt{Bel}_i\,\texttt{After}_{j:\alpha}\varphi \wedge \texttt{Bel}_i\,\texttt{Capable}_{j:\alpha} \rightarrow \texttt{Possible}_i\texttt{F}\varphi$$

Once we trust someone to do an action to bring us something, we hope for the positive result of the action. In case of success, we feel satisfaction (formalized as Proposition 4). Conversely, in case of failure, we feel disappointment (formalized as Proposition 5).

**Proposition 4** *(Successful Trust implies Satisfaction)*

$$\texttt{Bel}_i\texttt{Done}_{j:\alpha}\texttt{Trust}(i,j,\alpha,\varphi) \wedge \texttt{Awareness}_i\varphi \rightarrow \texttt{Satisfaction}_i\varphi$$

This means that when we believe that what we trusted has now occurred, we are satisfied about it. For example, when the boss trusted his secretary to prepare the report ($\texttt{Done}_{secretary:prepare}\texttt{Trust}(boss, secretary, prepare, having\ report)$), and on the morning of the day after, he has received the report ($\texttt{Bel}_{boss}$ *having report*), then he is satisfied ($\texttt{Satisfaction}_{boss}$*having report*). This proposition has a corollary which is deduced from Proposition 1 and 4: When we experience that what we trusted has really occurred, we will also feel joy about it.

**Corollary 1**

$$\texttt{Bel}_i\texttt{Done}_{j:\alpha}\texttt{Trust}(i,j,\alpha,\varphi) \wedge \texttt{Awareness}_i\varphi \rightarrow \texttt{Joy}_i\varphi$$

**Proposition 5** *(Unsuccessful Trust implies Disappointment)*

$$\texttt{Bel}_i\texttt{Done}_{j:\alpha}\texttt{Trust}(i,j,\alpha,\varphi) \wedge \texttt{Awareness}_i\neg\varphi \rightarrow \texttt{Disappointment}_i\neg\varphi$$

This means that we feel disappointed if what we trusted does not in fact occur. For example, a businessman trusted his partner to arrive on time to negotiate a contract. The businessman feels disappointed if the partner has not yet arrived at the scheduled time.

**DisTrust and Fear.** Distrust features a negative expectation, involving fear of the other [LW00, AACS08]. We state the relation between *Distrust* based on ability and *Fear* as Proposition 6.

**Proposition 6** *(DisTrust implies Fear)*

$$\texttt{C-DisTrust}(i,j,\alpha,\varphi) \rightarrow \texttt{Fear}_i \neg\varphi$$

This means that if we distrust someone to do an action to bring us something then we fear that our desire might not be fulfilled. For example, the boss might distrust his assistant with the preparation of a report he needs, and more specifically distrusts him to finish the report by the next morning ($\texttt{DisTrust}(boss, assistant, finish, report)$). Therefore, he is fearful that he might miss the report the next morning ($\texttt{Fear}_{boss} \neg report$). This proposition will be proved by applying Lemma 2: if we believe that someone is unable to do an action to bring about something, then we believe that there is at least a future world without the expected result of this action.

**Lemma 2**

$$\texttt{Bel}_i \neg \texttt{After}_{j:\alpha} \varphi \rightarrow \texttt{Possible}_i \texttt{F} \neg\varphi$$

Once we distrust someone to do an action to bring about something, we experience fear. If the results are indeed negative, we feel fear-confirmed (formalized as Proposition 7). If, however the action is in fact successfully performed, we feel relief (formalized as Proposition 8).

**Proposition 7** *(Confirmation of DisTrust implies Fear-confirmed)*

$$\texttt{Bel}_i \texttt{Done}_{j:\alpha} \texttt{C-DisTrust}(i,j,\alpha,\varphi) \wedge \texttt{Awareness}_i \neg\varphi \rightarrow \texttt{FearConfirmed}_i \neg\varphi$$

If the boss realizes that his assistant really did not finish the report ($\texttt{Bel}_{boss} \neg report$), he feels fear-confirmed ($\texttt{FearConfirmed}_{boss} \neg report$). Combining the two Propositions 2 and 7, we arrive at a corollary: when we experience that what we distrusted has now happened, we feel distressed about it.

**Corollary 2**

$$\texttt{Bel}_i \texttt{Done}_{j:\alpha} \texttt{C-DisTrust}(i,j,\alpha,\varphi) \wedge \texttt{Awareness}_i \neg\varphi \rightarrow \texttt{Distress}_i \neg\varphi$$

**Proposition 8** *(Non-confirmation of DisTrust implies Relief)*

$$\texttt{Bel}_i \texttt{Done}_{j:\alpha} \texttt{C-DisTrust}(i,j,\alpha,\varphi) \wedge \texttt{Awareness}_i \varphi \rightarrow \texttt{Relief}_i \varphi$$

If the boss discovers that his assistant did in fact finish the report ($\texttt{Bel}_{boss}\, report$), he feels relieved ($\texttt{Relief}_{boss} report$).

## 5.2 Behavioral validation

Although the propositions that we proved in the previous section are intuitively plausible, some of them have not yet received behavioral validation from the field of experimental psychology. We decided to collect empirical data concerning three propositions in this article, related to the emotions that follow trust when it is confirmed (Proposition 4), and when it is unconfirmed (Proposition 5); and the emotions that follow distrust, when it is unconfirmed (Proposition 8) [1].

Following the analysis in (Section 4) which argues that trust is the conjunction of the intention, the capacity, and the agreement of trustee, the presence of *Agreement* is intentionally fixed for the future test. We therefore operationalize *Trust* as the conjunction of *Intention* and *Capacity*, and *Distrust* as the three remaining cases. Participants to the survey read 8 different stories, following a $2 \times 2 \times 2$ within-subject design. The variables manipulated in the stories were *Intention* (Yes/No), *Capacity* (Yes/No), and *Outcome* (Success/Failure). As an example, here is the story corresponding to *Intention* = *Yes*, *Capacity* = *Yes*, and *Outcome* = *Success*.

> Mr. Boss is the marketing director of a big company. He needs an important financial report before a meeting tomorrow morning, but he has no time to write it because of other priorities. He asks Mr. Support to prepare it and put it on his desk before tomorrow morning.
>
> - *Mr. Boss believes that Mr. Support has the intention to prepare the report in time.*
> - *Mr. Boss believes that Mr. Support is able to prepare the report in time.*
>
> The morning after, Mr. Boss finds the report on his desk when he arrives. In your opinion, what does he feel?

In the condition *Intention* = *No*, "Mr. Boss believes that Mr. Support has the intention to prepare the report in time" was replaced with "Mr. Boss believes that Mr. Support has no intention to prepare the report in time." In the condition *Capacity* = *No*, "Mr. Boss believes that Mr. Support is able to prepare the report in time" was replaced with "Mr. Boss believes that Mr. Support is unable to prepare the report in time." Finally, in the condition *Outcome* = *Failure*, "Mr. Boss finds the report on his desk when he arrives" was replaced with "Mr. Boss does not find the report on his desk when he arrives."

After reading each story, participants rated the extent to which the main character would feel each of 7 emotions, which included our target emotions, satisfaction, disappointment, and relief; but also some emotions that we included for exploratory purposes, such as anger or thankfulness. Ratings used a 6-point scale anchored at *Not at all* and *Totally*.

A total of 100 participants took part in an online survey. The survey was offered in two languages, French (30% of the final sample) and Vietnamese (70%). Language was entered as a control variable in all statistical analyses, but added only a small overall main effect on participants' responses, and will not be discussed any further.

Descriptive statistics are displayed in Table 1. Participants' responses were analyzed by means of a repeated-measure analysis of variance, aimed at detecting statistically reliable effects of Trust and Outcome on our emotions of interest.

[1] We could not test Proposition 7 for a linguistic reason: Neither in French nor in Vietnamese (the two languages used in our experiment) could we find an everyday term equivalent to 'fear confirmed'.

| | Satisfaction | | Relief | | Disappointment | |
|---|---|---|---|---|---|---|
| | Trust | Distrust | Trust | Distrust | Trust | Distrust |
| Success | 4.9 (1.5) | 4.6 (1.6) | 2.8 (1.9) | 3.6 (1.9) | 1.1 (0.6) | 1.3 (0.8) |
| Failure | 1.1 (0.5) | 1.4 (1.0) | 1.3 (1.0) | 1.3 (0.9) | 4.6 (1.7) | 3.2 (1.4) |

Table 1: Mean and standard deviations of affective ratings, as a function of Trust and Outcome.

**Satisfaction.** Unsurprisingly, the analysis of variance detected a huge effect of Outcome, $F(1,98) = 597$, $p < .001$, accounting for most of the observed variance, $\eta_p^2 = .86$. In other terms, Satisfaction is almost perfectly predicted by Outcome alone. The analysis, however, also detects a comparatively small interaction effect Outcome $\times$ Trust, $F(1,98) = 8.8$, $p < .01$, $\eta_p^2 = .08$, reflecting the fact that success is even more pleasant in case of trust. Table 1 shows that the biggest score of *Satisfaction* is in the case of Trust follows a Failure: $M = 4.9$, $SD < 1.5$. The data are in line with what was expected from Proposition 4.

**Relief.** The analysis detected main effects of Trust, $F(1,98) = 19.1$, $p < .001$, $\eta_p^2 = .23$; and Outcome, $F(1,98) = 127$, $p < .001$, $\eta_p^2 = .80$. However, these main effects were qualified by an interaction effect Trust $\times$ Outcome, $F(1,98) = 12.3$, $p < .001$, $\eta_p^2 = .31$. Table 1 shows that the score of *Relief* is especially high in the case of success is obtained despite of distrust: $M = 3.6$, $SD < 1.9$. This interaction reflects our expectation (Proposition 8).

**Disappointment.** The analysis detected main effects of Trust, $F(1,98) = 28.4$, $p < .001$, $\eta_p^2 = .16$; and Outcome, $F(1,98) = 389$, $p < .001$, $\eta_p^2 = .56$. However, these main effects were qualified by an interaction effect Trust $\times$ Outcome, $F(1,98) = 44.7$, $p < .001$, $\eta_p^2 = .11$. Table 1 shows that the score of *Disappointment* is especially high in the case of failure is obtained despite of trust: $M = 4.6$, $SD < 1.7$. This interaction reflects our expectation (Proposition 5).

## 6 Conclusion

This paper introduced a logical framework that can represent emotions, trust, and the formal relations between them. In other terms, it enables to analyze the transformation of trust (and distrust) into emotions. Furthermore, this logical framework fully respects the instantaneity of emotions that previous logics of emotions did not fulfill. Finally, the formal relations between emotion and trust laid bare by the logical framework were subjected to a behavioral validation following the methods of experimental psychology. The success of this behavioral validation gives strong support to our approach, which is shown to capture lay users' intuitions about trust-related emotion.

Although we have added time factor into almost emotional formulas, which enables to eliminate rightly emotion when the relevant event has passed a long time, but it have not yet helped us to represent the nature of continuous intensity of emotions. Additionally, this paper has formalized only the effect of trust/distrust on emotions but not yet the effect of emotions on trust/distrust. These current limitations are also the potential perspective for our future research.

**Acknowledgements:** This work has been supported by the Agence Nationale de la Recherche

(ANR), contract No. ANR-08-CORD-005-1, and by a doctoral scholarship awarded by the University of Toulouse, contract No. 26977-2007.

## Bibliography

[AACS08] P. Aghion, Y. Algan, P. Cahuc, A. Shleifer. Regulation and Distrust. *SUS.DIV–CEPR–PSEConference of Models of Cultural Dynamics and Diversity*, 2008.

[AHL09] C. Adam, A. Herzig, D. Longin. A logical formalization of the OCC theory of emotions. *Synthese* 168(2):201–248, 2009.

[Bry07] H. J. Bryce. Formalizing Civic Engagement: NGOs and the Concepts of Trust, Structure, and Order in the Public Policy Process. *Workshop on Building Trust Through Civic Engagement and for the International Political Science Association, Section on Governance, conference on Government Crisis in Comparative Perspective, Seoul, Korea*, 2007.

[CES86] E. M. Clarke, E. A. Emerson, A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems* 8(2):244–263, 1986.

[CF01] C. Castelfranchi, R. Falcone. Social Trust: A Cognitive Approach. In Castelfranchi and Tan (eds.), *Trust and Deception in Virtual Societies*. Pp. 55–90. Kluwer Academic Publishers, Dordrecht, 2001.

[CFL08] C. Castelfranchi, R. Falcone, E. Lorini. A non-reductionist Approach to Trust. In Goldbeck (ed.), *Computing with Social Trust*. Pp. 45–72. Springer, Berlin, 2008.

[CL90] P. R. Cohen, H. J. Levesque. Intention is choice with commitment. *Artificial Intelligence* 42:213–261, 1990.

[Dre95] F. Dretske. *Naturalizing the mind*. MIT Press, Cambridge, 1995.

[FL79] M. Fischer, R. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences* 18(2):194–211, 1979.

[Fri86] N. H. Frijda. *The Emotions: Studies in Emotion & Social Interaction*. Edition de la Maison des Sciences de l'Homme. Cambridge University Press, Paris, 1986.

[HKT00] D. Harel, D. Kozen, J. Tiuryn. *Dynamic Logic*. MIT Press, 2000.

[HL04] A. Herzig, D. Longin. C&L intention revisited. In *Proceedings of Int. Conf. of knowledge representation and reasoning KR'04*. Pp. 527–535. Morgan Kaufmann, 2004.

[HLH+08] A. Herzig, E. Lorini, J. F. Hübner, J. Ben-Naim, O. Boissier, C. Castelfranchi, R. Demolombe, D. Longin, L. Perrussel, L. Vercouter. Prolegomena for a logic of trust and reputation. In *Proceedings of 3rd International Workshop on Normative Multiagent Systems (NorMAS)*. Luxembourg, July 2008.

[Lah01] B. Lahno. On the Emotional Character of Trust. *Journal of Ethical Theory and Moral Practice* 4:171–189, 2001.

[Laz91] R. S. Lazarus. *Emotion & Adaptation*. Oxford University Press, 1991.

[LW00] R. Lewicki, C. Wiethoff. Trust, Trust Development, and Trust Repair. In Deutsch and Coleman (eds.), *The Handbook of Conflict Resolution: Theory and Practice*. Pp. 86–107. Jossey-Bass, San Francisco, CA, 2000.

[OCC88] A. Ortony, G. L. Clore, A. Collins. *The Congnitive Structure of Emotions*. The Cambridge University Press, 1988.

[Pri57] A. N. Prior. *Time and Modality*. Clarendon Press, Oxford, 1957.

[Rei09] R. Reisenzein. Emotions as metarepresentational states of mind: Naturalizing the belief-desire theory of emotion. *Cognitive Systems Research* 10(1):6–20, 2009.

[Sch01] K. R. Scherer. *Appraisal Processes in Emotion : Theory, Methods, Research*. Chapter Appraisal Considered as a Process of Multilevel Sequential Checking, pp. 92–120. Oxford University Press, New York, 2001.

[dS01] R. de Sousa. *The Rationality of Emotion*. MIT Press, 6 edition, 2001.

[SSJ01] K. R. Scherer, A. Schorr, T. Johnstone. *Appraisal Processes in Emotion: Theory, methode, Research*. Series in Affective Science. Oxford university Press, 2001.

# UI-Design Driven Model-Based Testing

[1]Judy Bowen and Steve Reeves

[1]University of Waikato, Hamilton, New Zealand

**Abstract:** Testing interactive systems is notoriously difficult. Not only do we need to ensure that the functionality of the developed system is correct with respect to the requirements and specifications, we also need to ensure that the user interface to the system is correct (enables a user to access the functionality correctly) and is usable. These different requirements of interactive system testing are not easily combined within a single testing strategy. We investigate the use of models of interactive systems, which have been derived from design artefacts, as the basis for generating tests for an implemented system. We give a model-based method for testing interactive systems which has low overhead in terms of the models required and which enables testing of UI and system functionality from the perspective of user interaction.

**Keywords:** User interface, prototyping, formal methods, unit testing

## 1 Introduction

Testing of interactive systems is a difficult task. It requires that we test the system's functionality, the interactive behaviour of the system and that the user interface (UI) is usable and aesthetically acceptable for users. As UIs become more complex and software applications become ubiquitous, often relying on new, and sometimes novel, modes of interaction, this difficulty increases. Testing for UIs is often confined to human-based usability testing which is used primarily to ensure users are able to understand and successfully interact with the system under test (SUT) and to measure qualitative responses to aesthetic considerations. While usability testing is an important activity, it is known to be time-consuming and costly and is therefore more successful when performed on systems which have already been well tested.

It is not always practical to separate the testing of system functionality from the UI, and relying on usability testing for interactive elements as well as usability increases time and cost as well as putting a heavier burden on the process in terms of the number and types of errors we rely on it to catch.

In any testing process generating the tests is a critical activity as we want to ensure that the tests have as wide coverage as possible in order to find as many errors as possible, but at the same time we do not want the test generation process to be so onerous that the process becomes impractical due to the length of time it takes and the level of expertise required. In the case of interactive systems the difficulty is again increased as test generation requires knowledge of, and the ability to formally consider, both the underlying functionality and the interactive behaviours.

Model-based testing alleviates some of the problems of test generation in general by providing a formal basis for the tests as well as oracles to compare results against. It lends itself to automatic generation of tests via tool support (see [UL06] for a comprehensive discussion of this) which helps reduce both time and effort required. It also provides a way of generating repeatable

tests and gives confidence in the coverage of the testing. However, model-based testing methods for interactive systems are not yet widespread and have several challenges to overcome if they are to become so.

Choosing an appropriate model to describe the UI is one such issue. Paiva *et al.* [PFV07] for example, highlight and try to address this problem by combining the formal testing framework of a popular programming language, Spec# for the C# language [Spe], with UML. Their aim is to integrate the formality of Spec# with the visual familiarity of UML to develop abstract models of both functional and UI behavioural requirements which can be used to test for coverage and correctness. They have also chosen state-based models to work with, which is a common choice, but this then requires work to manage the complexity caused by the management of large numbers of states, for example by working with hierarchical models such as those proposed in [PTFV05]. Belli [Bel01, Bel03] extends this idea by using regular expressions to model sequences of user interactions as part of a fault-modelling technique. The exploration of all possible sequences is, however, necessarily large, and the overhead in creating the models not insubstantial.

Comprehensive research on model-based testing for interactive systems has been undertaken by Memon *et al*. (see for example [XM06], [Mem07], [YCM09] and [Mem09]). One of the fundamental concerns of this work is the development of the model to be used for testing. Their methods are based on creating a model of an existing implementation which is then used to develop tests of event and interaction sequences which can be used for regression testing as new functionality is added or the SUT is refactored. In contrast, we are investigating the use of a pre-implementation model of the interactive system which is derived from UI design artefacts and which is linked to a formal specification of the functionality of the system. We aim to find out if such a model can be successfully used to generate tests and provide an oracle to test if a subsequent implementation correctly instantiates the specified interactive system.

In previous work we have developed models for UIs [BR08a, Bow08] which are based upon design artefacts created as part of a user-centred design (UCD) process. In this paper we investigate whether we can use these models as the basis for model-based testing for interactive systems. We propose that this will provide several benefits. Firstly, the models themselves are lightweight and easy to produce as part of standard UCD processes. They use abstraction within the state-based models to avoid state-explosion problems and as such they do not lead to some of the problems associated with other UI models (high overhead of development, complexity of understanding *etc.*) Secondly, we use these models to link the UI and interactive behaviours to a formal system specification which provides a formal model of the entire system enabling us to derive tests which are comprehensive and cover all aspects of the SUT. Thirdly, using the models in this way not only increases the benefits that the use of such models provides, but also enables us to further support UCD techniques formally and test our system from the perspective of interactivity. The models describe both the intended design from the point of view of the UI designer (in conjunction with their informal artefacts such as prototypes) as well as a demonstration of correctness with respect to the overall system and the relationship between system and UI designs. So, we can use the models to derive tests for the properties which have been captured informally and formally within early designs.

Using UI designs as the basis for testing is an approach also taken in [ACE+06] but their work is used as the basis for a test-driven development approach for the UI and follows the approach

of complete separation of UI considerations from underlying functionality. We are concerned with the integration of UI and system behaviours once we are at the point of implementation, and our aim is to use the formal models of the UI to derive tests which ensure correctness of the integration.

The IEEE Software Engineering Body of Knowledge [96494] says:

> "Testing is an activity performed for evaluating product quality, and for improving it, by identifying defects and problems."

That is, the purpose of testing is to find errors: a successful test is one that finds an error. In model-based testing the model gives us a description of correct behaviour, so we use this to determine where incorrect behaviour occurs by looking for situations which violate the model. This means finding defects in the functionality and in the way we present that functionality to users, via the user interface. The testing we propose is dynamic, running the program to check behaviour under certain test cases, so it is a post-implementation activity. There are, of course, limitations to model-based testing; we are not guaranteed to find all errors. But by examining the underlying specification for expected behaviours as described in the model we hope to expose as many as possible before we move on to human-based usability testing, which can then focus on finding the sorts of errors which cannot be detected by other testing means.

## 2 Example System

The example we use throughout this paper is a calendar application called SimpleCalendar which is used to display a monthly view of a calendar with events which are assigned to a particular day. The user can view a calendar as a monthly view or as a single day view. They can add events to any given day, view the events of any given day and can also edit or delete those events. Based on these functional requirements a formal specification was developed using Z [13502]. Here we give only some of the relevant (to our exposition here) parts of that specification, namely the description of the system state along with descriptions of some of the operations as an example of how the system is specified. We omit, for brevity, the type definitions, axiomatic definitions and the rest of the operations.

The system state contains various observable values: a set *allevents* of events (each event containing a valid date and a title); the current day, month and year; and a set *vdates* which represents the dates currently visible in the application, which is in turn a subset of *allDates*, the set of all valid dates.

The *AddEvent* operation extends the set *allevents* in the *Calendar* state by adding to it the event given in the observation $i?$, the rest of the state remains unchanged. The *RemoveEvent* operation performs the reverse by removing the event given in $i?$ from the set of events.

The *ShowPreviousMonth* and *ShowNextMonth* operations increment or decrement the observation *currentMonth*, and depending on the initial value of *currentMonth* increment or decrement the *currentYear* observation when necessary (if we move forward a month from December or back a month from January).

*Calendar*

$$allevents : \mathbb{P}\, EVENT$$
$$currentMonth : MONTH$$
$$currentYear : \mathbb{N}$$
$$vdates : \mathbb{P}\, allDates$$

*AddEvent*

$$\Delta Calendar$$
$$i? : EVENT$$

---

$$allevents' = allevents \cup \{i?\}$$
$$currentMonth' = currentMonth$$
$$currentYear' = currentYear$$
$$vdates' = vdates$$

*RemoveEvent*

$$\Delta Calendar$$
$$i? : EVENT$$

---

$$allevents' = allevents \setminus \{i?\}$$
$$currentMonth' = currentMonth$$
$$currentYear' = currentYear$$
$$vdates' = vdates$$

*ShowPreviousMonth*

$$\Delta Calendar$$

---

$$allevents' = allevents$$
$$currentMonth > 1 \Rightarrow currentMonth' = currentMonth - 1 \land currentYear' = currentYear$$
$$currentMonth = 1 \Rightarrow currentMonth' = 12 \land currentYear' = currentYear - 1$$
$$vdates' = allDates \rhd (currentMonth' \,..\, currentMonth')$$

*ShowNextMonth*

$$\Delta Calendar$$

---

$$allevents' = allevents$$
$$currentMonth < 12 \Rightarrow currentMonth' = currentMonth + 1 \land currentYear' = currentYear$$
$$currentMonth = 12 \Rightarrow currentMonth' = 1 \land currentYear' = currentYear + 1$$
$$vdates' = allDates \rhd (currentMonth' \,..\, currentMonth')$$

A series of designs and prototypes of the UI for SimpleCalendar were developed following a user-centred design process. At the end of the design iterations the prototypes given in figures 1 and 2 were accepted as the basis for the application's UI.

We create a link between the formal specification of the system and the user interface design by creating presentation models and presentation and interaction models (PIMs) [BR06],

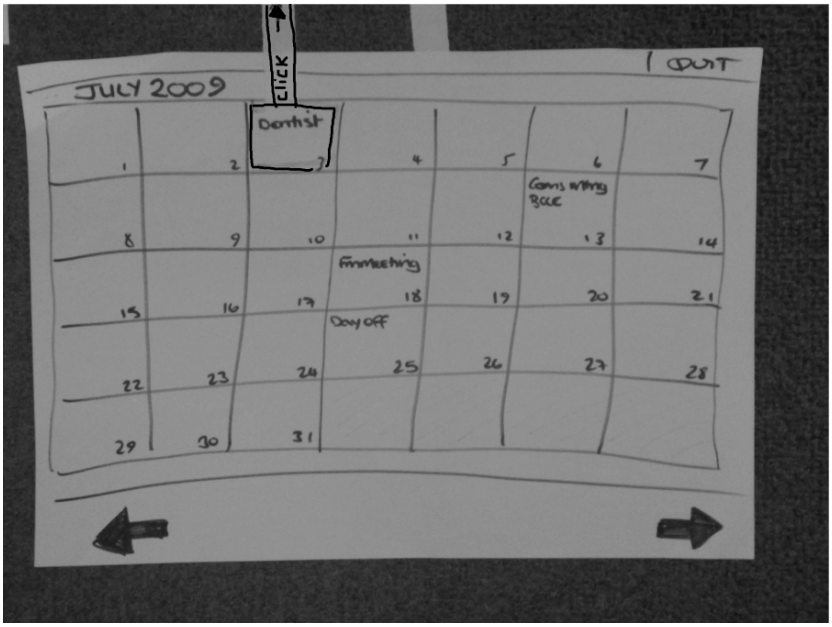


Figure 1: Main Month View for Simple Calendar

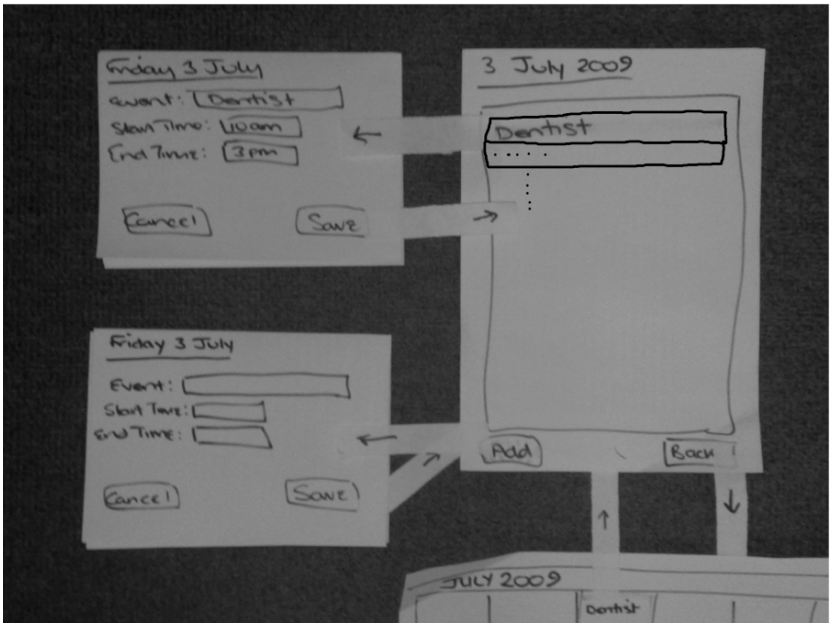


Figure 2: Subsidiary Views for Simple Calendar

[BR08a]. The presentation model gives a description of the interface designs based on the interactive elements (widgets) of the design. Each widget is described by way of a tuple consisting of a name, a category (which determines the type of interactive behaviour it exhibits) and a collection of behaviours associated with the widget. Behaviours either relate to system functionality (*i.e.* provide a way of interacting with the underlying system functionality) or to interface functionality, *e.g.* opening new dialogues, and are prefixed by S_ or I_ respectively. The UI for the entire system is described by a single presentation model which consists of component models for each of the distinct windows and dialogues. For the SimpleCalendar designs this is:

SimpleCal is MainView : DayView : AddView : EditView

MainView is
(QuitButton, ActionControl, (Quit))
(PrevArrow, ActionControl, (S_PrevMonth))
(NextArrow, ActionControl, (S_NextMonth))
(DayDisplay, ActionControl, (I_DayView))

DayView is
(AddButton, ActionControl, (I_AddView))
(EventList, ActionControl, (S_RemoveEvent, I_EditView))
(BackButton, ActionControl, (I_MainView))

AddView is
(TitleEntry, Entry, ())
(StartEntry, Entry, ())
(EndEntry, Entry, ())
(CancelButton, ActionControl, (I_DayView))
(SaveButton, ActionControl, (S_AddEvent, I_DayView))

EditView is
(TitleEntry, Entry, ())
(StartEntry, Entry, ())
(EndEntry, Entry, ())
(CancelButton, ActionControl, (I_DayView))
(SaveButton, ActionControl, (S_UpdateEvent, I_DayView))

We link the UI design models and the specification by creating a presentation model relation (PMR) between each S_Behaviour of the presentation model and operations of the specification, which for our example is *SimpleCal PMR*:

$$\{S\_PrevMonth \mapsto ShowPreviousMonth, S\_NextMonth \mapsto ShowNextMonth,$$
$$S\_RemoveEvent \mapsto DeleteEvent, S\_UpdateEvent \mapsto EditEvent, S\_AddEvent \mapsto AddEvent\}$$

The third model, the PIM, denotes the dynamic behaviour of the UI by describing how each individual dialogue or window is reached by way of I_Behaviours. Each component presentation model is associated with a state of the PIM, and I_Behaviours of the relevant model act as labels

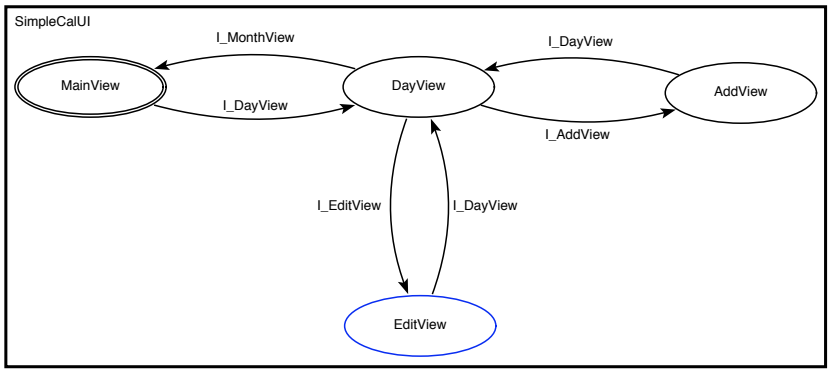


Figure 3: SimpleCal PIM

on transitions between states, and hence, as intended, are behaviours which are purely interface behaviours and so move us around the interface.

The combination of the system specification and the UI models (presentation models, PIM and PMR) provides a formal description of the entire system. We have previously shown how we can use this information as a way of ensuring correctness of the the proposed system [BR08a] and also as the basis for refinement [BR08b]. In this paper, however, we will use the models to derive tests which can then be run on an implementation of the system. The intention is that the models give a description of how we require the implemented system to behave and by using them to generate tests we hope to find errors where the implementation deviates from this behaviour. In the next section we show how the tests are derived.

## 3 Deriving the Tests

The presentation models describe the interactive elements of the UI and their required behaviours. That is, they describe the functionality that is accessible to a user who interacts with the UI. The PIM extends this to describe which behaviours are available in different states of the UI and how a user can move between these states. The testing approach we are proposing will ensure that both the behaviours, and the availability of the behaviours, are provided by the implementation so that we are sure that it satisfies the models. The PIM also describes modality: each independent state of the PIM is modal so we include this as a condition which should be tested.

UI-based testing is often goal-driven. Tasks are defined (or taken from earlier task analysis work) and then sequences of events and user interaction sequences are constructed to satisfy these goals (see for example [Bel01, WA00]). In contrast, the tests we derive use the definitions given within the models as their basis. These tests will be abstract (in that they are expressed at the level of, and in the language of, the models) and can then be instantiated in any language or using any testing framework as required. This will often be dependant on the choice of target implementation language. In section 4 we give an example of one way of instantiating the abstract tests for an implementation of SimpleCalendar in Java.

We begin by considering the dynamic behaviour of the UI. This is defined by I_Behaviours in the presentation models on transitions of the PIM showing how a user can move between states

of the UI. In the PIM given in figure 3 there are four states to be considered, with the initial state being *MainView* (denoted by the double ellipse). For all of the defined behaviours we will test two things: firstly that a widget exists in a given state which provides the required behaviour; and secondly that the behaviour is functionally correct. So, for example, the presentation model for *MainView* describes an *ActionControl* called *DayDisplay* which has a behaviour *I_DayView*. From the PIM we determine that this behaviour should cause the UI to change from the state *MainView* (*i.e.* a state where all of the defined behaviours of *MainView* are available) to the state *DayView* (*i.e.* a state where all of the defined behaviours of *DayView* are available). So first we will test that there is a widget available in *MainView* called *DayDisplay* and then we will ensure that when interaction occurs the UI behaves as required, that is it changes from *MainView* to *DayView*. During the testing process, in order to determine that we are in a correct state, we use the defined behaviours for that state. For example, the state *DayView* is a state of the UI where the behaviours of the *DayView* presentation model are available (a user has access to widgets with the behaviours $I\_AddView, S\_RemoveEvent, I\_EditView$ and $I\_MonthView$). The I_Behaviours and associated widgets for each of the states in our model are:

$$
\begin{aligned}
&MainView : \{DayDisplay \mapsto I\_DayView\}\\
&DayView : \{AddButton \mapsto I\_AddView, EventList \mapsto I\_EditView, BackButton \mapsto I\_MonthView\}\\
&AddView : \{CancelButton \mapsto I\_DayView, SaveButton \mapsto I\_DayView\}\\
&EditView : \{CancelButton \mapsto I\_DayView, SaveButton \mapsto I\_DayView\}
\end{aligned}
$$

Using this information we derive our first set of tests used to ensure that the relevant widgets exist in the appropriate states. To ensure that a widget is available for a user to interact with we must not only test that it exists in the given state, but also that it is visible and active. We describe the tests using first-order logic (which might be replaced by a table to show which predicates hold for which values in each state if that would be more suitable for various audiences) as follows:

$$
\begin{aligned}
&UIState(MainView) \Rightarrow Widget(DayDisplay) \wedge Visible(DayDisplay) \wedge Active(DayDisplay)\\
&\qquad \wedge hasBehaviour(DayDisplay, I\_DayView)\\
&UIState(DayView) \Rightarrow Widget(AddButton) \wedge Visible(AddButton) \wedge Active(AddButton)\\
&\qquad \wedge hasBehaviour(AddButton, I\_AddView)\\
&UIState(DayView) \Rightarrow Widget(EventList) \wedge Visible(EventList) \wedge Active(EventList)\\
&\qquad \wedge hasBehaviour(EventList, I\_EditView)\\
&UIState(DayView) \Rightarrow Widget(BackButton) \wedge Visible(BackButton) \wedge Active(BackButton)\\
&\qquad \wedge hasBehaviour(BackButton, I\_MainView)\\
&UIState(AddView) \Rightarrow Widget(CancelButton) \wedge Visible(CancelButton) \wedge Active(CancelButton)\\
&\qquad \wedge hasBehaviour(CancelButton, I\_DayView)\\
&UIState(AddView) \Rightarrow Widget(SaveButton) \wedge Visible(SaveButton) \wedge Active(SaveButton)\\
&\qquad \wedge hasBehaviour(SaveButton, I\_DayView)\\
&UIState(EditView) \Rightarrow Widget(CancelButton) \wedge Visible(CancelButton) \wedge Active(CancelButton)\\
&\qquad \wedge hasBehaviour(CancelButton, I\_DayView)\\
&UIState(EditView) \Rightarrow Widget(SaveButton) \wedge Visible(SaveButton) \wedge Active(SaveButton)\\
&\qquad \wedge hasBehaviour(SaveButton, I\_DayView)
\end{aligned}
$$

(The predicates here have the obvious (from their names) meaning, for now. They will be given a formal meaning by associating them with computed properties (via pieces of code) later on.) Next we ensure the modality of each state of the PIM (note that it is not necessary to put a

modality requirement on the initial state, *MainView*):

$$UIState(DayView) \Rightarrow Modal(DayView)$$
$$UIState(AddView) \Rightarrow Modal(AddView)$$
$$UIState(EditView) \Rightarrow Modal(EditView)$$

In order to derive tests for the system functionality we similarly identify the widgets with S_Behaviours and ensure that each of the widgets exist and that they have the required behaviours. When we come to instantiate the tests we can use the *PMR* to identify the specified operation which relates to the behaviour and then use the specification to determine the functionality which must be satisfied when the widget is interacted with. The functional tests we derive from the models are, therefore, as follows:

$$UIState(MainView) \Rightarrow Widget(QuitButton) \wedge Visible(QuitButton) \wedge Active(QuitButton) \wedge hasBehaviour(QuitButton, Quit)$$
$$UIState(MainView) \Rightarrow Widget(PrevArrow) \wedge Visible(PrevArrow) \wedge Active(PrevArrow) \wedge hasBehaviour(PrevArrow, S\_PrevMonth)$$
$$UIState(MainView) \Rightarrow Widget(NextArrow) \wedge Visible(NextArrow) \wedge Active(NextArrow) \wedge hasBehaviour(NextArrow, S\_NextMonth)$$
$$UIState(DayView) \Rightarrow Widget(EventList) \wedge Visible(EventList) \wedge Active(EventList) \wedge hasBehaviour(EventList, S\_RemoveEvent)$$
$$UIState(AddView) \Rightarrow Widget(SaveButton) \wedge Visible(SaveButton) \wedge Active(SaveButton) \wedge hasBehaviour(SaveButton, S\_AddEvent)$$
$$UIState(EditView) \Rightarrow Widget(SaveButton) \wedge Visible(SaveButton) \wedge Active(SaveButton) \wedge hasBehaviour(SaveButton, S\_UpdateEvent)$$

Finally we consider the widgets which do not have associated behaviours. In order for our implementation to satisfy the requirements given in the models we must also ensure that these non-functional widgets exist and can be seen by the user. Such widgets are used for a user to provide information to the system by way of inputs or to give information regarding the state of the system back to a user by way of displays.

$$UIState(AddView) \Rightarrow Widget(TitleEntry) \wedge Visible(TitleEntry) \wedge Active(TitleEntry)$$
$$UIState(AddView) \Rightarrow Widget(StartEntry) \wedge Visible(StartEntry) \wedge Active(StartEntry)$$
$$UIState(AddView) \Rightarrow Widget(EndEntry) \wedge Visible(EndEntry) \wedge Active(EndEntry)$$
$$UIState(EditView) \Rightarrow Widget(TitleEntry) \wedge Visible(TitleEntry) \wedge Active(TitleEntry)$$
$$UIState(EditView) \Rightarrow Widget(StartEntry) \wedge Visible(StartEntry) \wedge Active(StartEntry)$$
$$UIState(EditView) \Rightarrow Widget(EndEntry) \wedge Visible(EndEntry) \wedge Active(EndEntry)$$

This is the full set of abstract tests we derive from the models for the SimpleCalendar application. They define all of the conditions on an implementation. The tests provide coverage criteria, we know what we want to test and refer to the fixed properties of the UI which have been given initially within the UI design artefacts (the prototypes of figures 1 and 2 in this example). When we instantiate the tests we will see that we may need to define variables in some instances which are subject to the usual testing considerations of boundaries and choice of values. We discuss this further in the next section and show how we use the current visible state of the UI from a user's perspective to help with these choices.

In the next section we discuss how we instantiated the tests for a Java implementation of SimpleCalendar and give some positive and negative results of the testing process.

# 4 Instantiating and Running the Tests

Having shown how we can derive a set of abstract tests from formal models of UI design artefacts we now give an example of instantiating and running these tests. The Simple Calendar application has been implemented in Java and we have used the FEST testing framework [FES], which is based on the principles of TestNG and Abbot [RP07], as a way of instantiating and running the tests. While FEST is intended to provide a test-driven development approach to interactive system development, its ability to replicate user interaction (by way of the underlying Java Robot class) makes it a suitable approach for our work. It enables us to take a user-centred approach to our testing in the manner of replicating user interaction with the system to determine correctness of response to possible interaction with the UI and we can then use the underlying support of JUnit to determine whether or not the system behaves as described in our abstract tests. Due to the requirements of FEST classes, which rely on implementation details (such as widget names *etc.*), we take a white-box approach to testing where we use code inspection to determine the information required for FEST (as necessary).

Depending on *how* we want to test the system we might choose different ways of instantiating the tests. For example it may be enough to determine that all required behaviours of all UI states can be accessed by a user, or we might be stricter and require that if our model has two separate controls with a particular behaviour then the tests must show that two such distinct widgets exist with the required behaviour. This is the approach we have taken with this example as it adheres to our commitment to using the designs as the basis for implementation. That is, we expect everything described in the final design artefacts to become part of the implementation.

Just as we did when we began the test derivation process we start by considering the dynamic behaviour of I_Behaviours. In order to determine correctness of state we will ensure that each named state has the correct set of widgets visible to a user and available for interaction. FEST uses a package of classes called *Fixtures* which understand simulation of user events on Java Swing objects and verify the state of these objects. There are different classes for different types of widgets, for example a *JButtonFixture* enables simulation of clicks or double clicks *etc.* upon an actual JButton of an implementation (which is passed to the constructor of the fixture object). In order to test correctness of state, therefore, we create fixtures for each frame or dialogue which instantiates one of the states given in the PIM and then interrogate this to determine whether or not required widgets are present and correctly available. The following code is an example of such a test for the *MainView* state:

```
mv = new FrameFixture(new MView());
public void mViewState(){
  mv.button("quitButton").requireVisible();
  mv.button("quitButton").requireEnabled();
  mv.button("prevArrow").requireVisible();
  mv.button("prevArrow").requireEnabled();
  mv.button("nextArrow").requireVisible();
  mv.button("nextArrow").requireEnabled();
  mv.panel(testdate).requireVisible();
  mv.panel(testdate).requireEnabled();
}
```

where `MView` is the class in our implemented system which provides the UI elements for the *MainView* of the application. When we call the `mViewState()` method from within a JUnit test

method the `MView` frame is created and run in exactly the same way as if we had launched the SimpleCalendar application, and the cursor can be seen moving around the UI over each widget as it identifies it in the same manner as a user moving the mouse to hover over each of the widgets. If any of the tests fail (for example if one of the widgets cannot be found or does not have the required visibility property) we get the standard JUnit red failure bar along with an explanation of the cause of the test failure.

We create similar test methods for *DayView*, *AddView* and *EditView* and then use these as part of our I_Behaviour tests. We can either instantiate each abstract test individually, or combine two or more into a single test. For example we combine the modality requirement given in $UIState(DayView) \Rightarrow Modal(DayView)$ with the state test method for *DayView* by adding `dv.requireModal();` to the state test. In order to instantiate an abstract test such as:

$$UIState(MainView) \Rightarrow Widget(DayDisplay) \wedge Visible(DayDisplay) \\ \wedge Active(DayDisplay) \wedge hasBehaviour(DayDisplay, I\_DayView)$$

we determine from the PIM that a control called *DayView* should have the *I_DayView* behaviour which should change the state of the system from *MainView* to *DayView*. As part of the preparation for our tests we create a FrameFixture called *mv* which allows us to simulate interaction with the UI and take us to any of the other states as required for testing. For example the FEST code for the test given above is:

```
public void mvIDayViewTest(){
  DialogFixture dv = mv.panel(testdate).click().dialog(testdate);
  dViewState(dv);
}
```

This simulates a user clicking on a *dayDisplay* widget (a JPanel in our implementation) which opens a new dialogue, `dv`, and we then check that this has the defined *DayView* state. One way of identifying widgets using FEST is by using their name, and in SimpleCalendar we use the current date of each *DayDisplay* panel as the name's value. `Testdate` is a variable containing the current date (as the system always starts up displaying the current month this is a suitable choice for the test variable) and so represents the name of one of the JPanel widgets in *MainView*. This is an example of a test which requires a variable value (a date). Our choice of value for this is made based on what choices are available to a user when the system starts up, so we test based on the dates of the current month and iterate through each of the values that would be visible to the user. The range of the values chosen are then the limits of what a user has access to. We do not randomly test arbitrary dates or seek to test boundary values, such as 01/01/00, 12/12/99 *etc.* as these do not reflect choices the user can make in the current state.

We construct tests as described above for all of the I_Behaviours, and when we run them one at a time we discover our first error. The $dvIAddViewTest()$, which instantiates the abstract test:

$$UIState(DayView) \Rightarrow Widget(AddButton) \wedge Visible(AddButton) \wedge Active(AddButton) \\ \wedge hasBehaviour(AddButton, I\_AddView)$$

fails, producing the error:

```
java.lang.AssertionError: .. property'modal' expected <true> but was <false>
```

When the *aViewState* test is called to ensure that the resulting state after clicking the *Add* button is correct, the modality test fails. In the implementation of SimpleCalendar *AddView* has not been set as a modal dialogue and so the test fails and our error is discovered. Once we have corrected this problem all of the I_Behaviour tests are passed.

We next move onto the non-behavioural widgets, which enables us to test that the implemented UI for SimpleCalendar contains the required widgets for user entry and display. As there are no behaviours associated with these widgets we test them based on their category, so for the abstract test:

$$UIState(AddView) \Rightarrow Widget(TitleEntry) \wedge Visible(TitleEntry) \wedge Active(TitleEntry)$$

we identify the category of *TitleEntry* from the presentation model *Entry* and then instantiate the test by checking that the widget allows user entry (we do not need to test that the widget is visible and active in the state as we have already done this as part of our state tests). Using FEST we simulate the user entering some string into the text field and then test that the value of the text field is the entered string:

```
String tString = "Test Text";
av.textBox("titleEntry").enterText(tString);
av.textBox("titleEntry").requireText(tString);
```

It may seem strange to test the value of the `titleEntry` text box immediately after setting it, but the `enterText` instruction does not set the value of the text box, it merely attempts to interact with it in the same way a user would, by selecting it with the mouse and then entering the keystrokes required to produce the string. If the 'editable' property of the text box was set to false the `enterText` instruction would be carried out (by way of mouse movement and keyboard input) but the textBox would not contain the required string and so the assertion would fail. Each of the non-behavioural widgets are tested in this manner and all of the tests are passed.

Finally we move onto the S_Behaviour widget tests. In order to create these we need to identify and simulate user action on each of the widgets in each state in the same manner as for the I_Behaviours, and use the specified behaviour of operations related via the *PMR* to determine whether or not behaviour is correct. As an example consider the abstract test:

$$\begin{aligned} UIState(MainView) \Rightarrow\ & Widget(PrevArrow) \wedge Visible(PrevArrow) \\ & \wedge Active(PrevArrow) \wedge hasBehaviour(PrevArrow, S\_PrevMonth) \end{aligned}$$

Just as we have done with the other widgets we need to ensure that the widgets are available and visible in the required UI state and that the behaviour is correct. In the case of the S_Behaviours the meaning is given by the specified operation, *ShowPreviousMonth* (described in section 2) which the S_Behaviour is related to via the *PMR*. Because the S_Behaviours enable the user to access the system functionality (and therefore change the system state) as part of our test we should ensure that whenever a user can perform such an operation (*i.e* when a widget with that behaviour is available for interaction) the pre-condition of the related operation holds. This ensures that we do not expose users to the possibility of putting the system into an unexpected state. Secondly we must test that the post-condition given by the invariant in the operation description holds after the interaction, *i.e.* that the correct operation has occurred and has left the system in the expected state. In the example we present in this paper the specification of

the system is given in Z [13502] and we use standard conventions for determining pre- and post-conditions for operations. However, it is not a requirement that Z is used, only that related operations can be identified within the given specification and then appropriate methods used to identify the requirements for testing the system state.

For the *PrevArrow* widget in the *MainView* UI state our test then entails the following steps:

- ensure the *PrevArrow* widget exists in the *MainView* state
- ensure the *PrevArrow* widget is visible and enabled in the *MainView* state
- ensure that the pre-condition of the *ShowPreviousMonth* operation holds in *MainView*
- ensure that the post-condition of the *ShowPreviousMonth* operation holds in *MainView* after interaction with the *DayView* widget

The pre-condition of the operation schema can be calculated using standard Z techniques, and can be simplified to $currentMonth = 1 \lor currentMonth \in 2\,..\,12$, which for the UI means testing that the displayed month is either January, or between February and December. The post-condition of the operation requires that we check the value of *allevents* is unchanged and that the visible dates are correctly determined by the new value of *currentMonth* which should be the month prior to the original value. For the FEST testing we are only interested in the UI elements, and therefore separate the non-UI requirements (in this case the condition on *allevents*) into a separate test which can be run using JUnit independently of UI elements. This leads to the following test:

```
public void mvSPrevMonthTest(){
  int cm = cal.get(Calendar.MONTH);
  int year = cal.get(Calendar.YEAR);
  String yearstring = Integer.toString(year);
  mv.label("monthLabel").requireText(makeMonth(cm));
  int pcm = cm -1;
  for(int i = 0; i < 12; i++){
      if(pcm == 11)
          yearstring = Integer.toString(year-1);
      if(pcm == 0)
          pcm = 12;
      String prevdate = makeMonth(pcm);
      mv.button("prevArrow").click();
      mv.label("monthLabel").requireText(prevdate);
      mv.label("yearLabel").requireText(yearstring);
      pcm --;
  }
  sysPostConditionPrevMonth();
}
```

The line `mv.label("monthLabel").requireText(makeMonth(cm));` checks the pre-condition by ensuring that the value of the month label is one of the values given by the `makeMonth()` utility method in the test class (which returns only values in the range January to December). The test runs through a twelve month cycle which ensures coverage of both possible post-condition cases irrespective of the start month. Finally the method `sysPostConditionPrevMonth()` is called which is the unit test for the non-UI parts of the post-condition.

As we work our way through the tests for the S_Behaviours we obtain an unexpected result for one of the tests. When we run the test instantiating the abstract test:

$$UIState(DayView) \Rightarrow Widget(EventList) \wedge Visible(EventList) \\ \wedge Active(EventList) \wedge hasBehaviour(EventList, S\_RemoveEvent)$$

we observe the simulated interaction, and conclude that the test should fail. We have created an event titled "Dentist" for a given date, and then test that after S_RemoveEvent this event is no longer displayed in *DayView* or *MonthView*. What we observe upon closure of the *DayView* dialogue is that the event is still displayed in *MonthView*.The test, however, which checks the value of the label displaying event values in *MainView* is passed, as is the JUnit test of the underlying system state which determines that the event has been successfully removed from the collection of events maintained by the system. The error is caused by a lack of graphics refresh by the Java Virtual Machine and so although the event has been correctly removed, and the label text reset to empty, the previous value remains on the screen. Given that it is possible to run all of the tests we created in the background and generate a report of any errors that occur it is quite possible that such an error could be missed by this form of testing. It is a reminder of the importance of performing usability testing with people at the conclusion of model-based testing where such an error would be easily detected.

## 5 Conclusions

In this paper we have shown how our formal models of UI design artefacts can be used as the basis for model-based testing of interactive systems. We showed how it was possible to derive tests and oracles from the models which cover all of the behaviour captured by the UI designs and system specification. The tests are UI driven (as the models are based on UI designs), which reflects our desire to follow a UCD approach supported by formal methods.

We have given an example of how the abstract tests we derive can be instantiated and run against a Java implementation using the FEST framework in conjunction with JUnit. This enabled us to program tests for the implementation (in the nature of white-box testing) and run them to both observe the interaction produced as well as obtain the feedback from FEST and JUnit with respect to whether the tests were passed or not.

During the testing of our example SimpleCalendar application we discovered a modality error where the behaviour of the implementation did not match the oracle given by the model. We also discovered an example of an error which could not be caught by either FEST or JUnit. Our aim in performing model-based testing in this way is to find as many errors as possible prior to performing human-based usability testing. We want to discover as many functional and interaction errors as possible so that user testing can focus on usability and aesthetic issues.

Using the models enabled us to produce a range of abstract tests which covered all of the described interactive behaviours of the UI design models. Further we have shown one way of turning these abstract tests into an implemented test suite that can produce useful results. We believe that this initial investigation into using design models for this purpose has shown it to be a useful area of research to proceed with.

Our tool for creating, editing and storing presentation models and PIMs is currently being extended to support creation and exporting of abstract tests in the manner described in this paper. This will remove the necessity to manually create the abstract tests and may also be able to support partial generation of concrete tests for particular testing strategies. For example we could

automatically generate test method stubs for Java to support the example given in this paper, or use other suitable extensions to the tool depending on how the tests are to be implemented. This seems feasible given the uniform way tests and their predicates are given semantics by code.

We are also interested in investigating this testing strategy further and looking at different ways of instantiating the tests. In particular we would be interested to discover whether alternative methods of instantiation lead to better, or worse, results than we obtained using FEST and JUnit. Given that FEST is intended to be used within a test-driven development (TDD) process we believe it is possible to perform TDD of interactive systems based on the same abstract tests as we have presented here. That is we would use the UI designs as the basis of unit-tests (both for the UI and functionality of the system) and then follow the usual TDD approach of implementing the system with the objective of passing the tests.

Finally we also plan to investigate the use of the the abstract tests presented here as the basis for usability testing. There are many *ad hoc* approaches taken to deciding how a system should be tested with users and we are interested to see if these model-driven tests provide a useful basis for such decisions, and what, if any, differences this leads to in terms of results when compared with task-driven approaches to usability testing.

## Bibliography


[13502] I. 13568. *Information Technology—Z Formal Specification Notation—Syntax, Type System and Semantics*. Prentice-Hall International series in computer science. ISO/IEC, first edition, 2002.

[96494] I. 9646-1. *Information Technology—Open Systems Interconnection—Conformance Testing Methodology and Framework, Part 1: General Concepts*. International Standards Organisation. ISO/IEC, first edition, 1994.

[ACE+06] M. Alles, D. Crosby, C. Erickson, B. Harleton, M. Marsiglia, G. Pattison, C. Stienstra. Presenter First: Organizing Complex GUI Applications for Test-Driven Development. *AGILE Conference* 0:276–288, 2006.

[Bel01] F. Belli. Finite-State Testing and Analysis of Graphical User Interfaces. In *ISSRE '01: Proceedings of the 12th International Symposium on Software Reliability Engineering (ISSRE'01)*. Pp. 34–43. IEEE Computer Society, Washington, DC, USA, 2001.

[Bel03] F. Belli. A Holistic View for Finite-State Modeling and Testing of User Interactions. 2003. Technical Report 2003/1, Institute for Electrical Engineering and Information Technology, The University of Paderborn, April 2003.

[Bow08] J. Bowen. *Formal Models and Refinement for Graphical User Interface Design*. PhD thesis, University of Waikato, Department of Computer Science, 2008.

[BR06] J. Bowen, S. Reeves. Formal Models for Informal GUI Designs. In *1st International Workshop on Formal Methods for Interactive Systems, Macau SAR China, 31 October 2006*. Volume 183, pp. 57–72. Electronic Notes in Theoretical Computer Science, Elsevier, 2006.

[BR08a] J. Bowen, S. Reeves. Formal Models for User Interface design artefacts. *Innovations in Systems and Software Engineering* 4(2):125–141, 2008.

[BR08b] J. Bowen, S. Reeves. Refinement for User Interface Designs. *Electronic Notes Theoretical Computer Science* 208:5–22, 2008.

[FES] FEST (Fixtures for Easy Software Testing). http://fest.easytesting.org/wiki/pmwiki.php

[Mem07] A. M. Memon. An event-flow model of GUI-based applications for testing. *Software Testing Verification and Reliability* 17(3):137–157, 2007.

[Mem09] A. M. Memon. Using Reverse Engineering for Automated Usability Evaluation of GUI-Based Applications. In *Software Engineering Models, Patterns and Architectures for HCI*. Springer-Verlag London Ltd, 2009.

[PFV07] A. Paiva, J. C. P. Faria, R. F. A. M. Vidal. Towards the Integration of Visual and Formal Models for GUI Testing. *Electronic Notes Theoretical Computer Science* 190(2):99–111, 2007.

[PTFV05] A. Paiva, N. Tillmann, J. Faria, R. Vidal. Modeling and testing hierarchical GUIs. In *D. Beauquier, E. Borger, and A. Slissenko, editors, ASM05*. Universite de Paris, 2005.

[RP07] A. Ruiz, Y. W. Price. Test-Driven GUI Development with TestNG and Abbot. *IEEE Software* 24(3):51–57, 2007.

[Spe] Spec #. Microsoft technical pages for Spec #:. http://research.microsoft.com/specsharp/

[UL06] M. Utting, B. Legeard. *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.

[WA00] L. White, H. Almezen. Generating Test Cases for GUI Responsibilities Using Complete Interaction Sequences. In *ISSRE '00: Proceedings of the 11th International Symposium on Software Reliability Engineering*. P. 110. IEEE Computer Society, Washington, DC, USA, 2000.

[XM06] Q. Xie, A. M. Memon. Model-Based Testing of Community-Driven Open-Source GUI Applications. In *ICSM '06: Proceedings of the 22nd IEEE International Conference on Software Maintenance*. Pp. 145–154. IEEE Computer Society, Washington, DC, USA, 2006.

[YCM09] X. Yuan, M. B. Cohen, A. M. Memon. Towards Dynamic Adaptive Automated Test Generation for Graphical User Interfaces. In *ICSTW '09: Proceedings of the IEEE International Conference on Software Testing, Verification, and Validation Workshops*. Pp. 263–266. IEEE Computer Society, Washington, DC, USA, 2009.

# Towards the Verification of Pervasive Systems

**Myrto Arapinis[a], Muffy Calder[b], Louise Denis[c], Michael Fisher[d], Philip Gray[e], Savas Konur[f], Alice Miller[g], Eike Ritter[h], Mark Ryan[i], Sven Schewe[j], Chris Unsworth[k], Rehana Yasmin[l],**

[a] m.d.arapinis@cs.bham.ac.uk [h] e.ritter@cs.bham.ac.uk
[i] m.d.ryan@cs.bham.ac.uk [l] r.yasmin@cs.bham.ac.uk
School of Computer Science, University of Birmingham

[c] l.a.denis@liverpool.ac.uk [d] mfisher@liverpool.ac.uk
[f] s.konur@liverpool.ac.uk [j] sven.schewe@liverpool.ac.uk
Department of Computer Science, University of Liverpool

[b] muffy@dcs.gla.ac.uk [e] pdg@dcs.gla.ac.uk
[g] alice@dcs.gla.ac.uk [k] chrisu@dcs.gla.ac.uk
Departent of Computer Science, University of Glasgow

**Abstract:** Pervasive systems, that is roughly speaking systems that can interact with their environment, are increasingly common. In such systems, there are many dimensions to assess: security and reliability, safety and liveness, real-time response, etc. So far modelling and formalizing attempts have been very piecemeal approaches. This paper describes our analysis of a pervasive case study (MATCH, a homecare application) and our proposal for formal (particularly verification) approaches. Our goal is to see to what extent current state of the art formal methods are capable of coping with the verification demand introduced by pervasive systems, and to point out their limitations.

**Keywords:** pervasive systems, modelling, formalizing, verification

## 1 Introduction

*Pervasive systems* (often also termed *ubiquitous systems*) are increasingly common. But what are they? One of the many definitions is that

> *Pervasive Computing refers to a general class of mobile systems that can sense their physical environment, i.e., their context of use, and adapt their behaviour accordingly [PSHC08].*

While there are very many forms of pervasive system, they are often:

- mobile and autonomous;
- distributed and concurrent;
- interacting and interactive;

- reactive and potentially non-terminating; and
- composed of humans, agents and artifacts interacting together.

Existing pervasive systems provide services to the inhabitants of a home, the workers of an office or the drivers in a car park. We know that requirements for current and future pervasive systems involve great diversity in terms of their types of services, such as multimedia, communication or automation services.

**Typical Example.** *As we walk into a shopping area, our intelligent clothing interacts wirelessly with shops in the area and then with our mobile phone to let us know that our shoes are wearing out and that the best deals nearby are at shops 'X', 'Y' and 'Z'.*

*Our PDA, which holds our shopping list, also interacts with our phone to suggest the optimum route to include shoes in our shopping.*

*At the same time, our PDA interacts with the shopping area's network and finds that one of our friends is also shopping – a text message is sent to the friend's mobile/PDA to coordinate shopping plans and schedule a meeting for coffee at a close location in 15 minutes.*

Even in this simple example the components at least need capabilities to carry out:

- plan synchronisation;
- spatial reasoning and context-awareness;
- planning and scheduling;
- mobility and communication, etc.

The above is *an* example but there are *many* more, often more complex examples[1]. What is of concern to us is that pervasive computing is increasingly used in (safety, business, or mission) critical areas. This, of course, leads us to the potential use of formal methods in this area. Again, even considering the simple example above there are many dimensions to assess: security and reliability; safety and liveness; real-time response, etc.

Although there have been some attempts at modelling, e.g. [CFJ03, WZGP04, HI04, SB05, Sim07], formalising and even verifying aspects of pervasive and ubiquitous systems, e.g. [DKNP06], these have been very piecemeal approaches. We wish to be able to analyse the varied behaviours of a pervasive system from a number of viewpoints.

But how might this be achieved? This paper describes our analysis of a pervasive case study, MATCH [CM07], and our proposals for formal (particularly verification) approaches.

As we have seen, pervasive systems have some quite complex specification aspects. This explains, in part, why the verification of such systems is difficult. Essentially, most pervasive systems involve many dimensions that we must address (formalise and verify) simultaneously:

[1] See *Personal and Ubiquitous Computing* journal (Springer); *Pervasive and Mobile Computing* journal (Elsevier); *Journal of Ubiquitous Computing and Intelligence* (American Scientific Publishers); *International Journal of Ad Hoc and Ubiquitous Computing* (Inderscience Publishers); and *Journal of Ubiquitous Computing and Communication* (UBICC publishers).

- autonomous behaviour of agents;
- uncertainty in communications;
- teamwork, collaboration and coordination;
- organisations, norms, societal interactions;
- uncertainty in sensing;
- real-time aspects;
- *etc*...

In addition, such systems often involve humans *within* the system which directly affect the system's behaviour.

Current state of the art formal methods appear incapable of coping with the verification demand introduced by pervasive systems, primarily because reasoning about such systems requires considering quantitative, continuous and stochastic behaviour. We also require to prove interaction properties which are quite subtle to express.

## 2 MATCH overview

MATCH (Mobilising Advanced Technology for Care at Home) is a collaborative research project focused on technologies for care at home. The overall aim of MATCH is to develop a research base for advanced technologies in support of social and health care at home. The client users for MATCH will include older people and people with disabilities of all ages. The goal is to enable people to manage their health and way of life so that they can continue to live independently in their own homes for longer.

The main aim of the research in the MATCH project is to integrate a number of home-care technologies into one system that can be installed into a user's home. This system will provide support and assistance where needed by realising a number of goals. Goals may include "Warn the user if they are doing something dangerous", " Call for help if the user has an accident and needs assistance", "Inform a medical professional if the user's health deteriorates".

This could be implemented as a rule based, event driven pervasive system in which a set of input components such as: motion detectors, RFID readers, temperature sensors, heart monitors, and microphones, enable the detection of significant events. A set of output components such as: speakers, GUIs, mobile devices, TVs, lights, and tactile devices, allow the system to interact with the environment. A set of rules will determine under what circumstances the system should take action and what action should be taken.

A significant factor in whether such a system would be adopted within a home would be the issue of trust. Therefore, it would be advantageous if properties of the MATCH system could be verified. This would then provide support to the claims made by the system developers. Properties of interest can be categorised as (but not limited to) Security, Safety and/or Usability.

## 2.1 Security

Security properties relate to the integrity of the system and its ability to withstand the efforts of malicious agents. As an example the property "Food delivery staff cannot see mental health records" is concerned with the user's confidentiality. It would be advantageous if the food delivery staff could have limited access to the homecare system. This could be used to find relevant information about the user, such as dietary requirements. However, it would not be reasonable for all of the user's confidential medical records to be divulged to anyone who interacts with the system. Another example, "Medical records are consistent across the system" relates to system integrity. Because of the distributed nature of a pervasive system, there maybe several different devices used to monitor and record the medical condition of the user. A number of these devices may hold similar or overlapping data. It should be the case that all such data is consistent. "Correspondence between the system view of events and the events taking place on portable devices" is another integrity property. It is unlikely that portable devices will have complete knowledge of the state of the system. However, they should act in a manner that corresponds to the current state of the system as a whole. "No unauthorised tracking" is a property which is concerned with confidentiality. It should not be possible for a malicious agent to use system outputs to allow them to track the user.

## 2.2 Safety

Safety properties relate to situations in which the system may cause harm, either by action or inaction. For example, "Sensors are never offline when a patient is in danger". The system should always have sufficient sensor coverage to detect all potentially dangerous situations that it can be reasonably expected to recognise. If the system does detect that a patient is in danger then appropriate action should be taken. The action taken should have an expected response time that is appropriate for the situation. For example, if the patient is having a heart attack then an ambulance should be called. Other forms of notification such as e-mail are unlikely to be received in time. This could be represented by the property "If a patient is in danger, assistance should arrive within a given time". Properties such as "No component will perform an action that it believes will endanger the patient" can relate the hardware devices used within the system. A more traditional property, "Urgent actions related to the patient's safety will always take precedence over all other actions" ensures that important actions are prioritised. It is also important to consider potentially damaging actions a user may attempt, "Users have no reasonable strategy for cooperating to falsify records or events". This property aims to prevent users from being able to manipulate the system so as to deceive a health professional.

## 2.3 Usability

Usability properties refer to aspects that directly effect the user's experience and interaction with the system. Example properties include, "Notifications occur only at appropriate intervals and in appropriate circumstances". If a user is bombarded with too many insignificant messages or is interrupted while busy, they are likely to find the system obtrusive and will probably reject it. Similarly "Requests to the users must be relevant to them" and "There should not be to many pending requests on a user/patient" also relate to the users' acceptance of the system.

# 3 Some sample properties

In this section we will recall the earlier sample properties of the MATCH system, and formulate them using formal languages. In particular, we ideally consider formal languages for which a model checking or verification tool is available.

Below we consider an informal and formal account of the relevant properties.

## 3.1 Security

### 3.1.1 Formalisation

***Food delivery staff cannot see mental health records.*** In order to formalise this property we could use the access control language RW of [GRS04]. In that system, we assume predicates such as

$$\begin{array}{lcl} fd(X) & : & X \text{ is a member of food delivery staff} \\ mhr(X,Y) & : & X \text{ is the mental health record of } Y \\ td(X,Y) & : & X \text{ is a treating doctor for } Y \\ ha(X) & : & X \text{ is a health administrator} \end{array}$$

Predicates *read* and *write* indicate read and write permissions respectively. The following formulae show how such permissions are granted:

$$\begin{array}{lcl} read(mhr(X,Y),Z) & \Leftrightarrow & Y = Z \ \vee \ td(Z,Y) \\ write(td(X,Y),Z) & \Leftrightarrow & Z = Y \ \vee \ ha(Z) \\ & \vdots & \end{array}$$

RW can calculate whether a user can manipulate the access control system in order to give himself the necessary permissions to achieve a goal. In this example, a member of food delivery staff might be able to read mental health records if he can promote himself to treating doctor. We can verify that this is not possible by evaluating the query

$$\forall a,b, \in Agent.\ r \in Record.\ fd(a) \ \rightarrow \ \neg[a : mhr(r,b)]$$

This query evaluates whether there is a (perhaps roundabout) sequence of reads and writes resulting in a food delivery staff member $a$ being able to read the health records of some patient $b$.

***Correspondence between the system view of events and the events taking place on portable devices.*** This property is known as injective agreement in Lowe's hierarchy of authentication [Low97]. Intuitively, this property states that each time the MATCH system stores some value $v$ in the record of a patient $p$, then $p$'s doctor $d$ has submitted this value $v$. In the same way, each time doctor $d$ submits some value $v$ for $p$'s record to the system, then the MATCH system should update its state accordingly. This can be expressed in ProVerif's query language [Bla01] as follows:

$$\begin{array}{l} query\ evinj:\ ev_{syst}(d,p,v) \ ==> \ evinj:\ ev_{patient}(d,p,v) \\ query\ evinj:\ ev_{patient}(d,p,v) \ ==> \ evinj:\ ev_{syst}(d,p,v) \end{array}$$

The first (*resp.* second) query is true when, for each executed event $ev_{syst}(d,p,v)$ (*resp.* $ev_{patient}(d,p,v)$), there exists a distinct executed event $ev_{patient}(d,p,v)$ (*resp.* $ev_{syst}(d,p,v)$); and $ev_{syst}(d,p,v)$ is executed before $ev_{patient}(d,p,v)$ (*resp.* $ev_{patient}(d,p,v)$ is executed before $ev_{syst}(d,p,v)$).

***Non-authorised tracking* (e.g. *by strangers outside the house*).** MATCH's implementation will rely on Radio Frequency IDentification (RFID) tags attached to patients, in order for the system to be able to detect if a particular patient is in the house. However, one wouldn't want someone outside the house with an RF reader to be able to determine patient's position. In the RFID literature [GJP05, Jue06, WSRE03], this security requirement is known as untraceability. Intuitively, this property states that an attacker cannot link two different identifications to the system of the same tag (and thus of the same patient). In other words, all tags that identify themselves to the system look different to an outsider.

This property can be specified in the applied pi calculus formalism. The applied pi-calculus [AF01] is a language for describing concurrent processes and their interactions. It is based on the pi-calculus, but adds equations which make it possible to model a range of cryptographic primitives.

We consider RFID protocols that can be expressed in the applied pi calculus as a closed plain process $P$

$$P \equiv \nu\tilde{n}.\ (DB \mid !R \mid !T)$$

where

$$T \equiv \nu\tilde{m}.\ init.\ !main$$

for some processes *init* and *main*[2]. Intuitively, $T$ is the process modelling one tag, and having $T$ under a replication in $P$ corresponds to considering a system with an unbounded number of tags. Each tag initialises itself (this includes registering at the database $DB$ and is modelled by *init* in $T$) and then may execute itself an unbounded number of times. Thus *main* models one session of the tag's protocol. $R$ corresponds to one session of the reader's protocol, and $DB$ to the database. We consider an unbounded number of readers, thus $R$ is under a replication in $P$.

Many properties of security protocols (including untraceability) are formalised in terms of *observational equivalence* ($\approx$) between two processes. Intuitively, processes which are *observationally equivalent* cannot be distinguished by an outside observer, no matter what sort of test he makes. This is formalised by saying that the processes are indistinguishable under any context, *i.e.* no matter in what environment they are executed.

Let $P$ be an RFID protocol as defined above, $P$ is said to satisfy untraceability if $P \approx P'$ where

$$P' \equiv \nu\tilde{n}.\ (DB \mid !R \mid !T')$$

and

$$T' \equiv \nu\tilde{m}.\ init.\ main$$

The intuitive idea behind this definition is as follows: each session of $P$ should look to the intruder as initiated by a different tag. In other words, an *ideal* version of the protocol, *w.r.t.* untraceability, would allow tags to execute themselves at most once. The intruder should then not be able to tell the difference between the protocol $P$ and the ideal version of $P$.

[2] $\nu\tilde{n}$ models a sequence of names $n_1,\ldots,n_k$ restricted to $(Db \mid !R \mid !T)$. In the same way $\nu\tilde{m}$ denotes a sequence of names $m_1,\ldots,m_\ell$ restricted to $(init.\ !main)$.

### 3.1.2 Verification - at present

Several tools are available for verifying the properties defined above. Considering the first property, RW is supported by the AcPeg tool [Zha06] which accepts descriptions of access control models and evaluates queries. It treats the case that the sets of agents and other resources are finite (this case is decidable). It is currently not able to deal with non-bounded sets of resources, a problem known to be undecidable [HRU76].

DynPAL [Bec09] also analyses access control systems of a dynamic nature, similar to RW, and is more able to treat unbounded systems, but does not handle queries about "read" capabilities.

We expressed the second property in ProVerif's query language and the third using observational equivalence. This, in some cases, allows a user to automatically check that a protocol satisfies these requirements using the tool ProVerif [Bla01]. However, the verification of these properties is an undecidable problem [DLMS99], and ProVerif isn't always able to give an answer. It may even introduce false attacks.

For the second property, which is a correspondence property one could use other tools like Avispa [ABB+05] or Casper [Low98]. In order for these tools to be able to give some results, only bounded systems are considered. Indeed, when the number of executions of a protocol is bounded, the verification of many correspondence properties becomes decidable. To the best of our knowledge, ProVerif is the only tool able in some cases to prove that processes are observationally equivalent.

## 3.2 Safety

### 3.2.1 Formalisation

***Sensors are never offline when a patient is in danger.*** We can, for example, formulate such a statement using standard linear-time temporal logic (LTL) [Eme90, MP92, Fis07]. We first capture the notions of sensors being "offline" and patients being "in danger".

Assume $s_i$ is a sensor ($i \in \{1,..,m\}$). We define propositions[3] $\mathit{fail}(s_i)$, $\mathit{switch\_off}(s_i)$ and $\mathit{offline}(s_i)$ as follows: $\mathit{fail}(s_i)$ denotes that the sensor $s_i$ fails; $\mathit{switch\_off}(s_i)$ denotes that $s_i$ is switched off; and $\mathit{offline}(s_i)$ denotes that $s_i$ is offline. Now, we assume that if either the sensor fails or it has been switched off, then it is offline

$$\Box[(\mathit{fail}(s_i) \vee \mathit{switch\_off}(s_i)) \Rightarrow \mathit{offline}(s_i)], \quad \forall i \in \{1,..,m\}$$

(Recall that, in LTL, '$\Box$' means "at *all* present and future time points", '$\Diamond$' means "at *some* present or future time point", and '$\bigcirc$' means "at the *next* time point".) We now consider the cases where patients are in danger. Assume $p_j$ is a patient ($j \in \{1,..,n\}$). We define the following propositions: $\mathit{in\_danger}(p_j)$ denotes that the patient $p_j$ is in danger; $\mathit{high\_heart\_rate}(p_j)$ denotes that the heart rate monitor of $p_j$ has detected $p_j$'s heart rate to be higher than normal; $\mathit{low\_activity}(p_j)$ denotes that the activity monitors of the patient $p_j$ have detected that $p_j$ has moved very little for a period of time; $\mathit{motionless}(p_j)$ denotes that sensors have detected that $p_j$ has not moved at all for a period of time. If we assume that a patient is in danger only if either

[3] While we use a fragment of first-order language, the finiteness of the domain in question ensures that this is essentially *propositional* temporal logic.

his/her heart rate is higher than normal, he/she has moved very little with a period of time, or he/she has been inactive (but not in bed) for a while, then

$$\Box[(high\_heart\_rate(p_j) \vee low\_activity(p_j) \vee motionless(p_j)) \Rightarrow in\_danger(p_j)]$$

The truth of predicates such as '*low_activity*' can also be defined in terms of the values of sensors, together with some real-time and probabilistic constraints.

We can now specify the property "if a patient is in danger sensors never go offline until the patient is no longer in danger" as follows:

$$\Box[in\_danger(p_j) \Rightarrow (\neg \bigvee_{i\in\{1,..,m\}} offline(s_i))\mathscr{U}(\neg in\_danger(p_j))], \quad \forall j \in \{1,..,n\}$$

Of course, this simple version assumes that the dangerous situation will eventually be resolved.

Let us now consider the second property.

***Urgent actions related to the patients' safety will always take place before other actions*** Assume $A$ is a set of *urgent actions*, and $B$ is a set of *non-urgent* actions such that $A \cap B = \emptyset$ (again, this notion of urgency might well be defined in terms of some priority measure.) We use the proposition $action(a)$ to denote that the action '$a$' takes place. Property (ii) states that if a patient is in danger, a non-urgent action should not take place before an urgent action. This can again be expressed in LTL as follows:

$$\Box[in\_danger(p_j) \Rightarrow (\neg(\bigvee_{b\in B} action(b)\ \ \mathscr{U}\ \bigvee_{a\in A} action(a))\mathscr{U}\neg in\_danger(p_j)], \quad \forall j \in \{1,..,n\}$$

This, of course, can be made more detailed and complex, for example if we delve into the properties of actions.

***If a patient is in danger, assistance should arrive within a given time*** We define the proposition $assistance(p_j)$ as denoting that assistance arrives for $p_j$. If the specification were "*if* a patient is in danger, assistance should arrive eventually", then we could formulate this in LTL as follows:

$$\Box[in\_danger(p_j) \Rightarrow \Diamond assistance(p_j)], \quad \forall j \in \{1,..,n\}.$$

assistance must be available. Since this is a real-time property, we must extend the logic to capture this specification. Alternatively, we might use a logic such as TCTL [ACD90]. TCTL is a branching-time temporal logic, specifically a real-time extension of the logic CTL [CE82], which can express real-time properties. Thus, (ii) can be expressed in TCTL as follows:

$$\forall\Box[in\_danger(p_j) \Rightarrow \forall\Diamond_{\leq t} assistance(p_j)], \quad \forall j \in \{1,..,n\}$$

This formula states that if a patient is in danger, it is guaranteed that some assistance will arrive within time $t$. (Note that '$\forall$' here refers to *all* possible paths through the branching futures.)

***If a patient is in danger, assistance should arrive within a given time with a probability of 95%*** This property has both real-time and probabilistic aspects. Therefore, TCTL can no longer be used. In order to express this specification we can use the logic PCTL [HJ94], which can express quantitative bounds on the probability of system evolutions. Thus, (iv) can be specified in PCTL as follows:

$$P_{\geq 0.95}[in\_danger(p_j)\ \mathscr{U}^{\leq t}\ assistance(p_j)], \quad \forall j \in \{1,..,n\}$$

***No component will take an action that it believes will endanger the patient*** The property (v) now includes a situation where each component's beliefs must be taken into account. This suggests the use of a modal logic which uses possible worlds to capture the beliefs (or knowledge) that the component/agent has. Temporal Logics of Knowledge or Belief[FHMV96, HMV04] can be used to express the property (v) as they combine the temporal aspects with a modal logic of knowledge (or belief). Assume $c_i$ ($i \in \{1,..,C\}$) is a component agent, and $A$ is the set of all urgent and non-urgent actions. (v) might be expressed in a temporal logic of belief as:

$$\Box K_{c_i}[\neg(\bigvee_{a\in A} action(a) \Rightarrow \Diamond \bigvee_{j\in\{1,..,n\}} in\_danger(p_j))], \quad \forall i \in \{1,..,C\}$$

### 3.2.2 Verification Approaches

There are various verification and model checking tools for the logics we introduced in the previous section. The properties *(i)-(v)* can be verified using a suitable model checker or verification tool described below.

For the logic LTL some well-known tools are NuSMV [CCGR99], SPIN [Hol03], VIS [Gro96], TRP++ [HK03]. NuSMV is an extension of the model checking tool SMV [McM93], which is a software tool for the formal verification of finite state systems. Unlike SMV, NuSMV provides facility for LTL model checking. Spin is an LTL model checking system, supporting all correctness requirements expressible in LTL, but it can also be used as an efficient on-the-fly verifier for more basic safety and liveness properties. VIS is a symbolic model checker supporting LTL. VIS is able to synthesise finite state systems and/or verify properties of such systems, which have been specified hierarchically as a collection of interacting finite state machines. TRP++ is a resolution based theorem prover for LTL. TRP++ is based on resolution method for LTL [FDP01].

The best-known tools for the logic TCTL are UPPAAL [BLL+95] and KRONOS [BDM+98]. UPPAAL is an integrated tool environment for modelling, validation and verification of real-time systems modelled as networks of timed automata, extended with data types. The tool can be used for the automatic verification of safety and bounded liveness properties of real-time systems. KRONOS is a tool developed with the aim to verify complex real-time systems. In KRONOS, components of real-time systems are modelled by timed automata and the correctness requirements are expressed in the real-time temporal logic TCTL.

PRISM [HKNP06] and APMC [HLMP04] are tools which can be used to model check PCTL formulae. PRISM is a probabilistic model checker, a tool for formal modelling and analysis of systems which exhibit random or probabilistic behaviour. It supports three types of probabilistic models: discrete-time Markov chains (DTMCs), continuous-time Markov chains (CTMCs) and Markov decision processes (MDPs), plus extensions of these models with costs and rewards. The

property specification language is mainly PCTL; however, the tool also supports the temporal logics CSL and LTL. The "Approximate Probabilistic Model Checker" (APMC) [HLMP04] is an approximate distributed model checker for fully probabilistic systems. APMC uses a randomised algorithm to approximate the probability that a temporal formula is true, by using sampling of execution paths of the system.

Finally, there are tools for tackling the verification of specifications in combined modal and temporal logics, such as temporal logics of belief or temporal logics of knowledge. Although less well developed that some of the tools above, these range from deductive approaches [JHS+96, DFB02] to model checking for such logics [KLP04, GM04, BDFF08].

# 4 Discussion

## 4.1 Limitations

In Section 3, we give some example properties of different dimensions, e.g. security and safety, and describe various tools that can be used in the verification and model checking of these properties. There are several limitations of these tools and techniques.

As discussed in Section 3.2.1 and Section 3.2.2, different formal frameworks and tools are used to specify and verify different safety properties. For example, we should consider different temporal logics and model checking tools for real-time aspects, probabilistic aspects, belief aspects, etc. Unfortunately, there is no standard methods which can be used for all aspects we are interested in. This makes it difficult to use a certain formal framework for all safety properties. Another limitation is that each tool requires a different presentation of the model of the system.

As we discussed in Section 3.1.2, there exist several tools that allow us to verify systems *w.r.t.* security. However, some security properties of interest in pervasive systems are not supported well or even at all by these tools. Indeed, while ProVerif (to name only one) is very efficient for verifying correspondence and reachability properties like injective agreement, it seems to reach its limits when used for verifying observational equivalence, and thus properties like untraceability. This is due to the fact that ProVerif considers a stronger relation than observational equivalence introducing in practise many false attacks.

All the existing tools for verifying systems *w.r.t.* security abstract away from real time, focusing only on the sequencing of events. Although this has many advantages, it is a serious limitation for reasoning about protocols such as distance bounding protocols, which rely on real time considerations. These protocols are often implemented in RFID-based systems, and thus in many pervasive applications, in order to prevent relay attacks. Modelling time will thus be necessary for reasoning about pervasive systems.

In the same way, some features of many protocols designed to achieve the above mentioned requirements are not efficiently handled by existing tools. In particular, ProVerif is also inefficient for systems with local non-monotonic mutable states. But, protocols which aim to enforce untraceability often rely on such states. More precisely, ProVerif handles all states as monotonic introducing again false attacks.

As far as tools for verifying access control systems are concerned, the biggest limitation would be, as we already mentioned in section 3.1.2 that they usually cannot deal with unbounded access control models. Moreover, it is difficult to express and model integrity constraints in these tools.

Finally, we have introduced three existing theories for formally specifying security properties, namely RW, ProVerif's query language, and observational equivalence. It is important to note that they involve different levels of abstraction. Indeed, one can reason at the access control policy level, or at some symbolic model for protocols level, or even at the implementation level. It will be very important, to link these levels in order to transfer results from more abstract levels to more concrete ones.

### 4.2 Our Approach

As mentioned, a pervasive system might have quite complex specification aspects. Most pervasive systems involve many dimensions that must be formalised and verified simultaneously. Some of the dimensions that are common in pervasive systems are *real-time aspects*, *uncertainty in sensing and communication*, *teamwork, collaboration and coordination*, *organizations, norms and social interactions*, *autonomous behaviour of agents*, *etc.*

Current state of the art of formal methods appears incapable of coping with the verification demand introduced by pervasive systems, because reasoning about such systems requires combinations of multiple dimensions such as quantitative, continuous and stochastic behaviour to be considered, and requires proving properties which are quite subtle to express. For example, [Zah09, KZF09] show that some simple properties of a typical pervasive system can be verified using a single verification tool; but a single verification approach cannot be used in verification of more complex properties involving different dimensions.

Generally speaking, while the formal description of pervasive systems is essentially multi-dimensional, we generally do not have verification tools for all the appropriate combinations. It is very clear that developing a framework covering all these dimensions is almost impossible, or verification over very complex frameworks is very challenging.

In order to tackle the challenge of pervasive system verification, we aim to combine the power of established verification techniques, notably *model checking*, *deduction*, *abstraction*, etc. In particular, we are currently working on a generic framework for the model-checking of combined logics, including logics of knowledge, logics of context, real-time temporal logics, probabilistic temporal logics, etc. This work is based on the work of [FMD04], where a framework is given for the model-checking of combined modal temporal logics. This framework does not capture complex logics, which are quite essential in specifying different dimensions of pervasive systems. We therefore will extend this approach to provide a coherent framework for the formal analysis of pervasive systems. We will then apply the new technique to the verification of combined properties of a sample pervasive system, e.g. MATCH.

## 5 Conclusion

This paper describes our analysis of a pervasive case study, MATCH. As an initial step, we formally specify some safety and security properties of the MATCH system. We discuss to what extent current state of the art formal methods are capable of coping with the verification demand introduced by pervasive systems, and we point out their limitations. We also give an account of our proposal for formal verification of pervasive systems.

## Bibliography

[ABB+05] A. Armando, D. A. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cullar, P. H. Drielsma, P.-C. Ham, O. Kouchnarenko, J. Mantovani, S. Mdersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Vigan, L. Vigneron. The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications. In Etessami and Rajamani (eds.), *CAV*. Lecture Notes in Computer Science 3576, pp. 281–285. Springer, 2005.

[ACD90] R. Alur, C. Courcoubetis, D. Dill. Model-Checking for Real-Time Systems. In *Proc. Logic in Computer Science (LICS)*. Pp. 414–425. 1990.

[AF01] M. Abadi, C. Fournet. Mobile values, new names, and secure communication. *SIGPLAN Not.* 36(3):104–115, 2001.
doi:http://doi.acm.org/10.1145/373243.360213

[BDFF08] R. H. Bordini, L. A. Dennis, B. Farwer, M. Fisher. Automated Verification of Multi-Agent Programs. In *Proc. 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. Pp. 69–78. 2008.

[BDM+98] M. Bozga, C. Daws, O. Maler, A. Olivero, StavrosTripakis, S. Yovine. Kronos: A Model-Checking Tool for Real-Time Systems. In *CAV '98: Proceedings of the 10th International Conference on Computer Aided Verification*. Pp. 546–550. Springer Verlag, 1998.

[Bec09] M. Y. Becker. Specification and Analysis of Dynamic Authorisation Policies. In *Proceedings of the 22nd IEEE Computer Security Foundations Symposium (CSF'09)*. IEEE Computer Society Press, Port Jefferson, NY, USA, jul 2009.
doi:10.1109/CSF.2009.9

[Bla01] B. Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *CSFW '01: Proceedings of the 14th IEEE workshop on Computer Security Foundations*. P. 82. IEEE Computer Society, Washington, DC, USA, 2001.

[BLL+95] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, W. Yi. UPPAAL — a Tool Suite for Automatic Verification of Real–Time Systems. In *Proceedings of Workshop on Verification and Control of Hybrid Systems III*. Lecture Notes in Computer Science 1066, pp. 232–243. Springer Verlag, 1995.

[CCGR99] A. Cimatti, E. Clarke, F. Giunchiglia, M. Roveri. NuSMV: A New Symbolic Model Verifier. In *Proceedings of International Conference on Computer-Aided Verification (CAV'99)*. Pp. 495–499. 1999.

[CE82] E. M. Clarke, E. A. Emerson. Using Branching Time Temporal Logic to Synthesise Synchronisation Skeletons. *Science of Computer Programming* 2:241–266, 1982.

[CFJ03] H. Chen, T. Finin, A. Joshi. An Ontology for Context-aware Pervasive Computing Environments. *Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review*, 2003.

[CM07] J. S. Clark, M. R. McGee-Lennon. MATCH: Mobilising Advanced Technologies for Care at Home. 2007. Poster at *Delivering Healthcare for the 21st Century*, Glasgow.

[DFB02] C. Dixon, M. Fisher, A. Bolotov. Resolution in a Logic of Rational Agency. *Artificial Intelligence* 139(1):47–89, July 2002.

[DKNP06] M. Duflot, M. Z. Kwiatkowska, G. Norman, D. Parker. A Formal Analysis of Bluetooth Device Discovery. *STTT* 8(6):621–632, 2006.

[DLMS99] N. A. Durgin, P. D. Lincoln, J. C. Mitchell, A. Scedrov. Undecidability of bounded security protocols. In *proceedings of the Workshop on Formal Methods and Security Protocols-FMSP*. 1999.

[Eme90] E. A. Emerson. Temporal and Modal Logic. In Leeuwen (ed.), *Handbook of Theoretical Computer Science*. Pp. 996–1072. Elsevier, 1990.

[FDP01] M. Fisher, C. Dixon, M. Peim. Clausal Temporal Resolution. *ACM Transactions on Computational Logic* 2(1):12–56, Jan. 2001.

[FHMV96] R. Fagin, J. Halpern, Y. Moses, M. Vardi. *Reasoning About Knowledge*. MIT Press, 1996.

[Fis07] M. Fisher. Temporal Representation and Reasoning. In van Harmelen et al. (eds.), *Handbook of Knowledge Representation*. Elsevier Press, 2007.

[FMD04] M. Franceschet, A. Montanari, M. De Rijke. Model Checking for Combined Logics with an Application to Mobile Systems. *Automated Software Engg.* 11(3):289–321, 2004.

[GJP05] S. L. Garfinkel, A. Juels, R. Pappu. RFID Privacy: An Overview of Problems and Proposed Solutions. *IEEE Security and Privacy* 3(3):34–43, 2005. doi:http://doi.ieeecomputersociety.org/10.1109/MSP.2005.78

[GM04] P. Gammie, R. van der Meyden. MCK: Model Checking the Logic of Knowledge. In *Proc. 16th International Conference on Computer Aided Verification (CAV)*. Lecture Notes in Computer Science 3114, pp. 479–483. Springer, 2004.

[Gro96] T. V. Group. VIS: A System for Verification and Synthesis. In *Proceedings of the 8th International Conference on Computer Aided Verification*. Pp. 428–432. 1996.

[GRS04] D. P. Guelev, M. Ryan, P. Y. Schobbens. Model-checking Access Control Policies. 3225:16 pages., 2004.

[HI04] K. Henricksen, J. Indulska. A Software Engineering Framework for Context-aware Pervasive Computing. In *Proceedings 2nd IEEE Conf. on Pervasive Computing and Communications*. Pp. 77–86. 2004.

[HJ94] H. Hansson, B. Jonsson. A Logic for Reasoning about Time and Reliability. *Formal Aspects of Computing* 6:102–111, 1994.

[HK03] U. Hustadt, B. Konev. TRP++ 2.0: A Temporal Resolution Prover. In *Proceedings of Conference on Automated Deduction (CADE-19)*. Pp. 274–278. 2003.

[HKNP06] A. Hinton, M. Kwiatkowska, G. Norman, D. Parker. PRISM: A Tool for Automatic Verification of Probabilistic Systems. In *Proc. 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06)*. Lecture Notes in Computer Science 3920, pp. 441–444. Springer, 2006.

[HLMP04] T. Hérault, R. Lassaigne, F. Magniette, S. Peyronnet. Approximate Probabilistic Model Checking. In *Proc. 5th International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI'04)*. Lecture Notes in Computer Science 2937. Springer, 2004.

[HMV04] J. Y. Halpern, R. van der Meyden, M. Y. Vardi. Complete Axiomatizations for Reasoning about Knowledge and Time. *SIAM J. Comput.* 33(3):674–703, 2004.

[Hol03] G. J. Holzmann. *The Spin Model Checker*. Addison-Wesley, 2003.

[HRU76] M. A. Harrison, W. L. Ruzzo, J. D. Ullman. Protection in Operating Systems. *Commun. ACM* 19(8):461–471, 1976.

[JHS+96] G. Jaeger, A. Heuerding, S. Schwendimann, F. Achermann, P. Balsiger, P. Brambilla, H. Zimmermann, M. Bianchi, K. Guggisberg, W. Heinle. LWB–The Logics Workbench 1.0. http://lwbwww.unibe.ch:8080/LWBinfo.html, 1996. University of Berne, Switzerland.

[Jue06] A. Juels. RFID security and privacy: a research survey. *IEEE Journal on Selected Areas in Communications* 24(2):381–394, 2006.

[KLP04] M. Kacprzak, A. Lomuscio, W. Penczek. From Bounded to Unbounded Model Checking for Temporal Epistemic Logic. *Fundam. Inform.* 63(2-3):221–240, 2004.

[KZF09] S. Konur, A. A. Zahrani, M. Fisher. Verification of a Message Forwarding System using PRISM. In *PreProc. of Ninth International Workshop on Automated Verification of Critical Systems*. Technical Report, University of Swansea, 2009.

[Low97] G. Lowe. A Hierarchy of Authentication Specifications. In *CSFW '97: Proceedings of the 10th IEEE workshop on Computer Security Foundations*. P. 31. IEEE Computer Society, Washington, DC, USA, 1997.

[Low98] G. Lowe. Casper: A Compiler for the Analysis of Security Protocols. *Journal of Computer Security* 6(1-2):53–84, 1998.

[McM93] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishing, 1993.

[MP92] Z. Manna, A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, New York, 1992.

[PSHC08] E. K. Paik, M.-K. Shin, J. Hwang, J. Choi. Design Goals and General Requirements for Future Network. N13490, Korea Technology Center, 2008.

[SB05] Q. Z. Sheng, B. Benatallah. ContextUML: A UML-based Modeling Language for Model-driven Development of Context-aware Web Services. In *Proceedings of the International Conference on Mobile Business (ICMB'05)*. 2005.

[Sim07] C. Simons. CMP: A UML Context Modeling Profile for Mobile Distributed Systems. In *Proceedings of the 40th Hawaii International Conference on System Sciences*. 2007.

[WSRE03] S. A. Weis, S. E. Sarma, R. L. Rivest, D. W. Engels. Security and Privacy Aspects of Low-Cost Radio. In *Hutter, D., Müller, G., Stephan, W., Ullman, M., eds.: International Conference on Security in Pervasive Computing - SPC 2003, volume 2802 of LNCS, Boppard, Germany*. Pp. 454–469. Springer-Verlag, march 2003.

[WZGP04] X. H. Wang, D. Q. Zhang, T. Gu, H. K. Pung. Ontology-based Context Modeling and Reasoning Using Owl. In *Context Modeling and Reasoning Workshop at PerCom 04*. Pp. 18–22. 2004.

[Zah09] A. A. Zahrani. Formal Analysis of a Message Forwarding System using PRISM. 2009.

[Zha06] N. Zhang. AcPeg, the access control policy evaluator and generator. July 2006. The tool can be obtained from www.cs.bham.ac.uk/ nxz or www.cs.bham.ac.uk/ mdr/research/projects/05-AccessControl.

# Tightly coupled verification of pervasive systems

**Muffy Calder, Phil Gray and Chris Unsworth**

Department of Computing Science
University of Glasgow Glasgow G12 8RZ, UK

**Abstract:** We consider the problem of verifying context-aware, pervasive, interactive systems when the interaction involves both system configuration and system use. Verification of configurable systems is more tightly coupled to design when the verification process involves reasoning about configurable formal models. The approach is illustrated with a case study: using the model checker SPIN [Hol03] and a SAT solver [ES03] to reason about a configurable model of an activity monitor from the MATCH homecare infrastructure [MG09]. Parts of the models are generated automatically from actual log files.

**Keywords:** formal models, pervasive systems, model checking

## 1 Introduction

Effective verification of interactive systems has been a significant challenge for both the verification and user interface communities for at least the last two decades [CH97, CRB07]. More recently, the advent of context-aware, pervasive, interactive systems raises the stakes: can we formulate effective verification techniques and strategies to bring reasoning into the design processes of these volatile systems? In particular, we are concerned with systems where the interaction involves both system configuration and system use.

Pervasive systems are characterised by their ability to sense their physical environment and a use of data so gathered both as part of the core application functionality and as a way of modifying system behaviour to reflect changes in the context of use. Amongst the challenges of designing, building and operating such systems is the volatility that this sensor-based context dependency introduces. The set of sensors may themselves come and go and change their behaviour depending upon environmental conditions. Context changes may be difficult to model and predict. In addition, pervasive applications often operate in situations that require practically unpredictable changes to the application functionality itself. For example, a home care system, providing sensor-based monitoring of the cared person's activities and state, may have to be reconfigured to take into account changes in the person's medical condition and their home situation, and consequent changes to the services and sensors needed. For these reasons, system configuration must be treated as an ongoing process throughout the lifetime of a system. It must be modelled and reasoned about in the same way that one would model and reason about normal user interaction with the system.

At a high level of abstraction these context-aware, interactive systems may be regarded as a number of concurrent processes:

$$Agents||Sensors||Outputs||Monitors||Configuration||System$$

where

- *Agents* are (usually, but not exclusively human) users, there may be several types and (possibly overlappping) instances of user, e.g. patient, carer, social worker, etc.
- *Sensors* and *Outputs* quantify physical world data (e.g. thermometer, pressure pad, web-cam), or are outputs devices (e.g. speaker, television screen),
- *Monitors* are high level abstractions of a physical state (e.g. encapsulating predicates about who is in a room, whether or not is it cold),
- *Configurations* are sets of rules, or actual parameters, determining how the system varies according to user preferences and needs,
- the *System* is the underlying computational infrastructure.

While we might traditionally consider the composition $(Agents||Sensors||Outputs||Monitors)$ as the context, i.e. together they reflect a temporal physical context, the $Configuration$ is also a context, in that it is also temporal and affects system behaviour.

In this paper we consider the modelling and verification process for configurable, interactive, context-aware systems in general, and a case study of an event driven system. We begin with a general purpose model of functional behaviour and for this we propose that a (concurrent) process based specification language, temporal logics and reasoning by model checking is a good paradigm, especially when context changes are non-deterministic. In the case study here we develop a general purpose model in Promela, for checking with the SPIN model checker [Hol03]. We then refine the verification problem and develop a specialised model for checking redundacies, using a SAT solver [ES03]. A distinctive feature of this work is parts of the models are generated automatically from actual log files.

In the next section we outline our overall vision for the modelling and verification process. The remainder of the paper is an exploration of one iteration of that process, for a case study. Section 3 introduces the MATCH case study and in section 4 we give an outline of our Promela model. Properties for verification are given in Section 5, where we give an overview of checking for redundant rules in the Promela model. In Section 6 we give an outline of the SAT model for redundancy checking and results of online verification. In Section 7 we consider the more general problem of overlapping left and/or right hand sides of rules, and when these should be interpreted as undesirable. Discussion follows in Section 8 and an overview of related work follows. Conclusions and future work are in Section 10.

## 2 Modelling and verification process

Traditionally, modelling is a manual process with the starting point of a system specification, or a set of requirements, or, when the system is operational, observations and data. One notable exception is [HS99], where the Promela model is extracted mechanically from call processing software code: a control-flow skeleton is created using a simple parser, and this skeleton is then populated with all message-passing operations from the original code. Our vision for the modelling and verification process is similar in that we aim to more tightly couple the model

and the system, and indeed the results of the verification. Crucial to the process is the notion of configuration and the extraction from the system of configuration details, often stored as log files.

Our vision is illustrated in Figure 1, where ellipses denote agents and rectangles objects. The key feature of our vision is that modelling is tightly coupled with system development and configuration. This is not a waterfall model: activities are concurrent and moreover, while four agent roles are indicated, they may be overlapping or conflated. Briefly, activities are as follows. The end users configure the system, and when configured, (possibly different) users interact with the system, as system and users require, according to the context. The configuration is not static, but may be changed repeatedly. Log files are a representation of the configuration process and are generated by a live system. The formal model depends upon what kind of analysis is required (e.g. functional behaviour, security, performance, etc.) and it is also configured, according to the log files. The model is analysed; the verification results may inform understandings of the end user, the configurer, the designer, and the modeller, though in different ways. For example, the user develops a better cognitive model, the configurer understands how to improve his/her rules, the designer develops a better interface, and the modeller gains insight in to how to modify the model so that verification is more efficient. Note, this is just an example. There may be multiple models and a single agent may have multiple roles as configurer/modeller/user/designer. Verification may be performed off-line or on-line, each of which has its merits. On-line verification can inform users in real-time. On the other hand, off-line verification allows more general results e.g. for all configurations with a certain property, a system proprety holds. This kind of verification can then be used by the designer to constrain allowable interactions or configurations. Finally, recall that agents may not be human at all, for example, the system might autonomously configure itself, or the modeller may be another software process.

Properties may support, for example,

- end user configurations, e.g. *what will happen if I add this rule?* or *how can I notify/detect x in the event of y?*
- modalities, e.g. *are there multiple speech outputs?* or *are there multiple speech inputs only when there are multiple users?*
- hypotheses about resources, e.g. *what happens if a webcam doesn't work?*

In this paper we report on one iteration of the modelling and verification cycle, starting with log files extracted from actual system trials of a prototype system (deployed in the UK and in France).

## 3 MATCH System

Activity awareness systems constitute an increasingly popular category of pervasive application [MRM09]. Such systems allow groups of users to share information about their current status or recent activities. They have a variety of purposes, ranging from supporting collaborative work through informal social relationships. We have chosen to investigate our approach to verification using one such activity awareness system, the MATCH Activity Monitor (hereafter, MAM), an

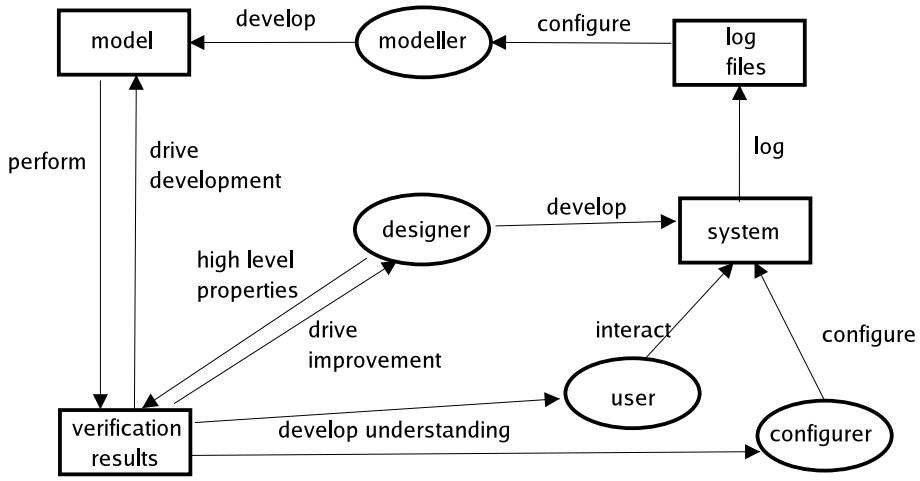


Figure 1: Tightly coupled verification: configurable systems and configurable models

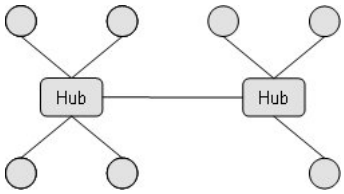


Figure 2: MAM System Architecture

experimental platform designed to explore the challenges of the configuration of activity awareness use to support of home-care [MG09].

A MAM system consists of one or more hubs (UMPC-based subsystems supporting a rich set of inputs and message types) each of which is connected to a set of satellites (web-based clients offering a limited set of inputs and message types) and other hubs, illustrated via the architecture diagram in Figure 2. Typically, a hub will reside in the home of a person requiring care while the satellites are used by carers, clinicians, family and friends.

Each hub, placed in the home of a cared person, can communicate with a set of web-based clients and with other hubs. A MAM hub supports a set of up to eight monitoring tasks, each of which involves the generation of messages based on user-generated or sensor-generated input indicating an event or activity. Monitoring tasks are defined by rules that specify an event or activity to be reported plus the destination and form of the reporting message. For example, a rule might state that use of Tony's coffee cup (captured via an appropriate sensor[1]. ) should be reported to me (i.e., the hub in my home) via a speech message. Currently, MAM supports a variety of data sources, message destinations and message modalities (e.g., speech, graphics,

[1] MAM uses a JAKE sensor pack for simple movement sensing, while the JAKEs more powerful sibling, the SHAKE, provides richer sensing capabilities and tactile feedback [JAK, SHA]

touch, etc.). A full list is given in Figure 3.

Each MAM hub can support up to eight monitoring tasks, each of which is specified explicitly as a monitoring rule[2]. In addition to simple <input source> <destination, modality> rules, it is also possible to specify combinations of inputs (e.g., a button press or an appointment) or message modalities (e.g., speech and graphics). Rules may also have a guard condition; currently MAM only supports a location condition such that the message is sent if someone is sensed near a specified location.

A user may also choose a system-generated recommendation of the input, destination or modality. The recommendation can be used in an automatic or semi-automatic mode. In the former case, the system will choose the input, destination or modality most commonly associated with the other parameters, based on a history of logged configurations. In the latter case, the system will offer a ranked list of choices, based again on frequency of association, from which the user must select one.

A user interface is supplied for specifying monitoring task rules. If a user is not interacting with the MAM Hub, it operates a digital photo frame application that displays the user's photos in order to make it a non-intrusive part of the user's home. To configure the hub, a user touches the screen and the photo application fades away, replaced by the MAM application, from which the configuration screen is accessible. Figure 4 shows a typical rule configuration. Note the eight tabs to the left, one for each rule; rule 1 is selected. The rule configuration view is divided into a left-hand panel for specifying input and a right-hand panel for destination and modality. In this case the blue and red buttons on my hub (left-hand panel) have been selected to create messages to be sent to Lucy's machine(s) (right-hand panel). The large vertical green button on the right of the panel is the on-off toggle switch for the rule; when green, the rule is active and when red the rule is inactive.

Even with the rather limited set of inputs, destinations and message types, the configuration space (i.e., number of different possible rules) is rather large (1.07E+301). In addition, not all configurations are desirable. It is possible to create redundant rules, which can be a problem given the restriction on the number of rules allowed. Also, some configurations may cause difficulties for the user: two speech messages delivered at the same time will be impossible to understand. These configuration challenges provide a motivation for verification that can be used both to guide a designer (in exploring the design of the configuration options offered to a user) and to help an end user (in creating a set of rules that both meet their needs and are understandable and maintainable).

## 4 General model

From a modelling perspective, the MAM system is an event driven rule-based pervasive system. Events include (but are not restricted to) direct user interaction with the hub, such as pushing buttons, and indirect user interaction such as movement captured by a webcam or external actions such as messages received from other users. Rules dictate how the system will react to events. We note that from a modelling perspective, there is no distinction between a user interaction and

[2] This limitation, amongst others, is intentional and based on empirical evidence, to limit the complexity of the application.

| Input Sources | |
|---|---|
| Calendar | An online calendar scheduling system reports upcoming appointments |
| Accelerometer | Small custom-built Bluetooth accelerometers can be placed around the home (e.g., on a phone, teacup, or door) or on a person, in order to detect movement-signalled activity of the instrumented thing/person. This is performed using JAKE and SHAKE devices [WMH07]. |
| Webcam movement events | Fixed and wireless webcams can be used to provide motion detection. This allows for room occupancy to be detected and reported. |
| User-generated text | Users can key in their current activity, mood or needs explicitly using an on-screen keyboard. |
| Abstract Buttons | A user may select an abstract button to which no particular meaning has been assigned in advance by the developers of the system (i.e. the red square) The user may negotiate with other people to assign a particular meaning to these buttons. This concept is derived from MarkerClock [RM07] that uses a similar abstract marker feature. |
| Message Destinations | |
| Local Hub | Messages are directed to one of the output devices associated with the local machine. |
| Registered Users | Messages are directed to specified users; the message will be sent to their hub, if they have one, or to their registered web-based client(s). |
| Modalities | |
| Graphical | Notice of an activity is briefly overlaid on top of the hub photoframe; an icon indicates that there is an unread message waiting. Additionally, the message will be added to a scrollable list of messages that is permanently available. |
| Speech | The content of the message is rendered into VoiceXML and played through any of the devices speakers. |
| Non-speech audio | A selection of auditory alerts is provided, such as nature sounds as well as more familiar alert noises. Each set of sounds contains multiple .wav files, each of which is mapped to a particular type of alert. As with speech, this can be directed to any distinct speaker. |
| Tactile | The SHAKE device (but not the JAKE) is equipped with an inbuilt vibrotactile actuator that can be activated. Vibration profiles (i.e. vibrate fast-slow-fast. slow-fast-slow) can be used to distinguish between different types of activity. |
| Email | Activity messages can be delivered to one or more email addresses that the user can specify. |

Figure 3: MAM Activity Monitoring Task Parameters

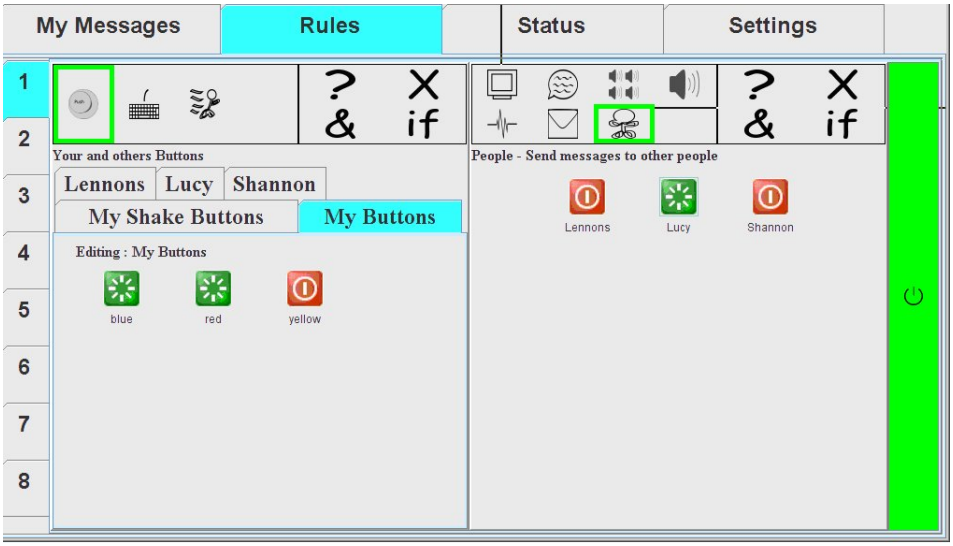


Figure 4: Sample task configuration screen

change of context. While there is intent associated with the former, from a modelling point of view both are simply aspects of state that may be captured by propositions (whose validity may be temporal).

Promela [Hol03] is a high-level, state-based, language for modelling communicating, concurrent processes. It is an imperative, C-like language with additional constructs for non-determinism, asynchronous and rendezvous (synchronizing) communication, dynamic process creation, and mobile connections, i.e. communication channels can be passed along other communication channels. The language is very expressive, and has intuitive Kripke structure semantics.

Our model is centred around a single hub that can take nput from one or more satellites or additional hubs. As a result we have a single rule set, which in this case is static. However, it would be a simple matter to extend the model to include multiple hubs and rule sets and dynamic rule sets.

As we are considering a configurable system, the model is designed to reflect this. The system behaviour is separated from the system configuration. System behaviour refers to the actions of the available input and output devices. System configuration refers to the current active rule set.

## 4.1 System

Each input device is represented by a global variable and a process. For example, a button press is represented by a single bit variable and a process that can arbitrarily assign the values 0 or 1.

Movement sensors such as a jake, shake or webcam, are represented as an integer variable. The movement process will arbitrarily assign a values 0-3, where 0 represents no movement and 1,2 and 3 represent low, medium and high levels of movement respectively. Text based inputs, such as messages from other hubs, are represented as an mtype variable. The associated process will arbitrarily assign values representing one of the users in the system or a null value. These

processes act as sources of events for the system.

Output devices act as sinks within the system. In the model they are represented as global variables or channels, upon which messages are placed. The associated process for an output device is called when a value is assigned to the variable/posted in the channel. The process then resets the variable or reads the message off the channel.

In future versions of the model it may be useful to include users and/or multiple hubs and rule sets in the system. In this case, the input variables will be directly modified by output processes from other hubs and/or as a direct result of a user action.

## 4.2 Rules

Rules are taken directly from a MAM system via the log. An excerpt from a log file can be seen in Figure 5, included as an illustration of the content of a log file (and not to be read in detail!).

```
uk.org.match_proj.osgi.EvalFunc.ManualEachComponentInputSelectionEvaluationFunction(selected=["Personal Messages"]){}
uk.org.match_proj.osgi.EvalFunc.UnionOutputApprovalEvaluationFunction()
{uk.org.match_proj.osgi.EvalFunc.ManualEachComponentOutputSelectionEvaluationFunction(selected=["GUI","Twitter"])
{}uk.org.match_proj.osgi.EvalFunc.ManualEachPersonSelectionEvaluationFunction(selected=["Doms","Lionel","Anne"]){}}
uk.org.match_proj.osgi.EvalFunc.ManualGroupedPersonSelectionEvaluationFunction(selected=["Everyone"]){}
uk.org.match_proj.osgi.EvalFunc.ManualEachComponentOutputSelectionEvaluationFunction(selected=["GUI"]){}
uk.org.match_proj.osgi.EvalFunc.ManualEachPersonSelectionEvaluationFunction(selected=["Caroline","Lionel","Anne"]){}
uk.org.match_proj.osgi.EvalFunc.ManualEachComponentOutputSelectionEvaluationFunction(selected=["Speech"]){}
uk.org.match_proj.osgi.EvalFunc.ManualEachComponentInputSelectionEvaluationFunction(selected=["Jake Movement"]){}
uk.org.match_proj.osgi.EvalFunc.UnionOutputApprovalEvaluationFunction()
{uk.org.match_proj.osgi.EvalFunc.ManualEachPersonSelectionEvaluationFunction(selected=["Caroline","Lionel","Anne"])
{}uk.org.match_proj.osgi.EvalFunc.ManualEachComponentOutputSelectionEvaluationFunction(selected=["GUI","Twitter"]){}}
uk.org.match_proj.osgi.EvalFunc.ManualEachComponentInputSelectionEvaluationFunction(selected=["Work Messages"]){}
uk.org.match_proj.osgi.EvalFunc.ManualEachPersonSelectionEvaluationFunction(selected=["Caroline","Anne"]){}
```

Figure 5: Excerpt from a MAM log file.

In the MAM system, rules are defined as evaluation functions that return input or output devices. Each rule consists of an input and an output evaluation function. The combination of function name and the list of parameters determine how they are to act. Both input and output functions can be composed. The input composition operator acts as a disjunction, meaning that an event will be triggered if either evaluation function is true. However, the output composition function acts as a conjunction, meaning that if the union output function is triggered then the result of both evaluation functions will be used.

The evaluation function rule set is then expressed as an informal natural language rule set. An example is shown in Figure 6. While this step is not strictly necessary, it can be helpful to have the rules expressed in a more readable format.

The rules are then expressed as Promela statements. Each rule is expressed as a conditional statement, consisting of a guard and a compound statement. Therefore, in Promela we represent a rule as a single statement $C \rightarrow A$, where $C$ is a guard statement made up of a disjunction of statements representing the condition of the rule and $A$ is compound statement consisting of a sequence of statements representing the action. For example, the rule "when the red console button is pressed play the doorbell earcon[3] on the hub speaker" maps to the Promela statement "$(this.red > 0) \rightarrow this.speaker!earcon\_doorbell$". Rules can also be context sensitive. For example, "If the red button is pressed then, if the webcam has recently detected movement inform

[3] An earcon is a short meaningful audio segment.

| | | |
|---|---|---|
| If the red or blue buttons are pressed | then | play the rocket earcon |
| If my webcam detects movement | then | display a pop-up message on my screen and display a message on the screen list |
| If I receive a message from Bill or Bill presses his red button | then | inform me using synthesised speech |
| If I receive a message from Bill | then | send a vibration message via the shake |
| If the red button is pressed | then | send a message to Bill and inform me using synthesised speech |
| If the shake senses movement | then | send a vibration message via the shake |
| If Bill presses his red button | then | inform me using synthesised speech |
| If the yellow button is pressed | then | send a vibration message via the shake |

Figure 6: Example rule set.

me with synthesised speech else send me an e-mail". In this case the definition of recent is a system parameter.

An example of a Promela representation of a rule set can be seen in Figure 7. In this example there are 2 hubs, one that belongs to the user and one that belongs to Bill. In the rule set the user's hub is referred to as *this* and Bill's hub is *Billh*. This is because in the model a hub is represented by a bespoke variable type.

```
proctype rules()
{
do
  :: (this.red == 1) || (this.blue == 1) -> this.audio_out!ec_rocket;
  :: (this.webcam > 2) -> this.screen_popup = me; this.screen_list = me;
  :: (this.text_in == Bill || Billh.red == 1) -> this.audio_out!speech;
  :: (this.text_in == Bill) -> this.shake_out = 1;
  :: (this.red > 0) ->  Billh.text_in = me; this.audio_out!speech;
  :: (this.shake_in_m > 1) -> this.shake_out = 1;
  :: (Billh.red > 0) -> this.audio_out = speech;
  :: (this.yellow > 0) -> this.shake_out = 1;
od
}
```

Figure 7: Promela representation of example rule set.

# 5 Properties

We now explore a number of issues in the MAM system that may benefit from formal verification.

## 5.1 Redundant rule detection

As the rules may be added by non-expert users, some may have overlapping or repeated definitions. It would be advantageous if the system could detect such redundant rules to be able

to streamline the system. This could provide feedback to the user and/or allow the system to remove redundant rules from the active rule set.

## 5.2 Modalities

The input and output devices can be classified by their modalities. For example, earcons and speech are sound, screen pop-ups and text messages are visual and vibration alerts are tactile. The acceptability of a system for a user may depend on the correct use of the different modalities. For example, multiple simultaneous audio outputs may confuse a user and result in the loss of messages. Visual output devices should be avoided for severely visually impaired users, however, it may still be appropriate to use them to notify carers. Overuse of tactile devices may result in the user being unable to differentiate between different types of messages.

## 5.3 Priorities

Currently MAM does not hold information on the relative priorities of rules/messages. However, a user is likely to be more concerned with certain messages than others. It would therefore be useful to check that the output from a given rule has a greater chance of being received by the appropriate target. For example, high priority messages should be distinct from lower priority messages. If a high priority message uses an earcon, then no other message should use a similar sounding earcon.

## 5.4 Verification results

We now expand on the redundant rule detection problem using the model checker SPIN, which verifies (or not) properties expressed in the logic LTL (linear temporal logic).

An LTL property can be derived easily from the Promela representation of a rule. In the general form, for a rule $r$ of form $C \rightarrow A$, where $C$ is a guard and $A$ is a sequence of statements, the associated property is informally described by: the action will always eventually occur after the condition becomes true (recall, conditions are disjunctions and actions are conjunctions. More formally, we define the mapping as $f(r) = \Box(f(C) \rightarrow \Diamond f(A))$, where $f()$ maps guards and assignments to propositions in the obvious way.

For example, the rule $(this.yellow > 0) -> this.shake\_out = 1$ would map to the LTL property $\Box((this.yellow > 0) \rightarrow \Diamond(this.shake\_out = 1))$.

We define $f(r)$ as the property that the action associated with rule $r$ will always eventually occur after the condition of $r$ becomes true. A rule $r$ in the rule set $R$ is redundant if for model $\mathscr{M}(R|r)$, which represents $R$ without $r$, $f(r) \models \mathscr{M}(R|r)$. A rule set $R$ contains no redundant rules if $\forall r f(r) \not\models \mathscr{M}(R|r)$.

Redundancy checking was implemented with a realistic rule set (shown in Figure 8) taken from an actual log file from the MAM system. Each rule was tested in-turn for redundancy. Rule r7 was found to be redundant. Verification times varied from around 12 minutes to 34 minutes. The search depth was between 4 and 6 million, and the number of states explored was between 65 and 100 million. Clearly if this is to be used for real-time verification then significant improvements need to be made in the model efficiency and/or the verification techniques employed. However,

these results do serve as a proof of concept.

| | | | |
|---|---|---|---|
| r1 | $((this.red == 1)\|\|(this.blue == 1)\|\|(this.yellow == 1))$ | $\rightarrow$ | $this.audio\_out = ec\_rocket$ |
| r2 | $(this.webcam > 2)$ | $\rightarrow$ | $this.screen\_popup = me$; $this.screen\_list = me$ |
| r3 | $(this.text\_in == Bill)\|\|(Billh.red == 1)\|\|(Billh.blue == 1)\|\|(Billh.yellow == 1)$ | $\rightarrow$ | $this.audio\_out = speech$; $this.screen\_list = me$ |
| r4 | $(this.text\_in == Bill)$ | $\rightarrow$ | $this.shake\_out = 1$ |
| r5 | $(this.red > 0)$ | $\rightarrow$ | $Billh.text\_in = me$ |
| r6 | $(this.shake\_in\_m > 1)$ | $\rightarrow$ | $this.shake\_out = 1$ |
| r7 | $(Billh.red > 0)$ | $\rightarrow$ | $this.audio\_out = speech$ |
| r8 | $(this.yellow > 0)$ | $\rightarrow$ | $this.shake\_out = 1$ |

Figure 8: Rule set used in experiments.

# 6 Specialised model

From Section 5.4 it can be seen that our current general Promela model of the MAM system is not sufficiently efficient to detect redundancy fast enough to provide feedback to a system configurer in real-time. In this section we show how redundancy can be modelled and solved efficiently with a specialised SAT solver [ES03]. SAT solvers check satisfiability of propositional formulae (usually written in disjunctive normal form). Though in general NP-complete, SAT solvers are highly efficient for many practical applications. The SAT model developed here uses literals to represent input devices, output devices and the rules.

## 6.1 Literals - inputs and outputs

Each simple input type is represented by a single literal. For example, a literal represents a button press or receiving a message from someone. Similarly, simple output types are also represented as literals. More complex input functions such as movement, which has low, medium and high inputs, can be represented as one literal per input value. A clause then needs to be added to ensure the input values are consistent. For example, if the literal for a movement level high is true, then both medium and low should also be true. To ensure this, the clauses $low \vee \neg medium$ and $medium \vee \neg high$ are added, which will need to be done for each movement detection device. However, if only one rule takes input from a movement detection device, then the input can be treated as a simple input device and the clause can be omitted. Classes of output, such as the MAM auditory icon class "nature", which consists of the auditory icons {wave, forest, wind}, can be modelled in one of two ways. If none of the individual auditory icons from this group are used as output for other rules, then the group can be represented as a single literal. Otherwise, each of the class members will be represented as a single literal and there is an additional literal for the class. For example, the class "nature" will be represented by $wave \vee forest \vee wind \vee \neg nature$

| | | | |
|---|---|---|---|
| 1. | $\forall r \in R$ | $\forall c \in r$ | $r_i \vee \neg c$ |
| 2. | $\forall r \in R$ | 1 | $c_1 \vee c_2 \vee \cdots \vee c_n \vee \neg r_i$ |
| 3. | $\forall r \in R$ | $\forall a \in r$ | $\neg r_i \vee a$ |
| 4. | $\forall a \in R$ | 1 | $r_1 \vee r_2 \vee \cdots \vee r_n \vee \neg a_i$ |

Figure 9: Clauses required to represent a given rule set $R$

### 6.2 Clauses - rules

Each rule is represented by a literal $r_i$ and a set of clauses as described in Figure 9. There are four types of clauses associated with a rule, as follows. The literal $r_i$ being assigned the value true indicates that rule $i$ has been triggered. A clause $r_i \vee \neg c$ is added for each condition $c$ that triggers rule $i$. To ensure $r_i$ is not true if none of its conditions are met, the following clause is added $c_1 \vee c_2 \vee \cdots \vee c_n \vee \neg r_i$. If rule $i$ is triggered then the appropriate outputs must be set to true, thus the clause $\neg r_i \vee a$ is added for each action $a$ associated with rule $i$. Finally, to ensure that actions are only taken if triggered by a rule, the clause $r_1 \vee r_2 \vee \cdots \vee r_n \vee \neg a_i$ is added, where $r_1$ to $r_n$ are all the rules that can trigger action $a_i$.

### 6.3 Rule redundancy

To use the above model to detect redundant rules, we need to solve the model once for each atomic condition in the rule, to check if atomic condition $c$ from rule $r_i$ is redundant. We add a clause for each input literal, setting the literal related to $c$ to true and all the rest to false. All literals that represent the actions from rule $r_i$ are set to true. All clauses related to the rule being checked are removed. If the resultant model is satisfiable then condition $c$ from rule $r_i$ is redundant, if all conditions associated with rule $r_i$ are redundant, then $r_i$ is redundant.

### 6.4 Implementation and complexity

The above model was implemented and solved with the same rule set used in Section 5.4. A Java program was written to read the rules from a file in the MAM evaluation function format and generate the SAT model. The SAT model was then solved using the open-source SAT solver *miniSAT* [ES03]. The problem had 23 literals and around 45 clauses[4], each instance of the problem required less than a thousandth of a second to solve. All instances were solved with propagation alone, no search was required.

This model was then used in conjunction with a Java program, which reads a rule set directly from a MAM system log file, generates the models and checks the rule set for redundancy. Using the actual hardware that MAM runs on, reading in and checking a rule set required approximately 5 seconds. The majority of this time was taken to read and parse the log file. Each individual SAT model required approximately 15 thousandths of a second to solve. It is clear that the specialised SAT model offers improvement of $5-6$ orders of magnitude for redundancy checking over the general Promela model.

[4] The number of clauses is dependant on the rule being checked.

## 7 Overlapping rules

We have defined a rule to be redundant if it can be removed from a system without affecting how the system operates. However, a rule may also be redundant if it serves no *useful* purpose. For example, a rule set may include the two rules "If I receive a message from Bill then play a bird noise earcon" and "If I receive a message from Bill then play a doorbell earcon". At a system level these rules are different. However, they both play a sound when a message is received from Bill.

One could interpret this as a lack of confluence, i.e. we have overlapping left hand sides of rules and divergent right hand sides. Rules can also overlap in more subtle ways (e.g. a form of superposition). For example, a rule set may include the two rules "If the red or yellow buttons are pressed play the doorbell earcon and send a text message to Bill" and "If the red or blue buttons are pressed play the doorbell earcon and inform me using synthesised speech". Both rules cause the doorbell earcon to be played, when one condition is satisfied.

While these are only simple examples, they raise the question of what exactly we should be looking for when detecting redundant rules and what is the underlying theory of modalities? Moreover, to answer these question we need to know why we are interested in this problem. For example, is it a significant issue in practice?

The answer to the latter appears to be positive. While conducting user evaluation studies, the developers of the MAM system have found that many test subjects indicated they have trouble understanding complex rules and only want to define simple rules. This means there is significant scope for overlapping conditions and actions. Therefore, it would be advantageous for the system to offer assistance. This could be in the form of a message informing the user that a rule they entered is redundant, or makes some other rule redundant, or is overlapping with another rule. Alternately, the system may simply detect such rules and only partially implement them. In any case, further study is need to understand user intentions and their relation to modalities and context, and also how best to feedback information from any analysis.

## 8 Discussion

A distinctive feature of this work is we are trying to more tightly couple design with modelling, closing the loop between design, use, configuration, modelling and verification. Further, we deal with systems as actually deployed, rather than an ideal yet to be implemented.

A long term goal is to automate many of these processes and so in this case study, where possible, we have developed scripts to process inputs automatically e.g. log files.

The general model is based around the central concept of event – the MAM system is after all event-driven. It captures a wide range of functional, temporal behaviour. However, in the context of checking for redundancy within rules, complexity of the model became a concern, especially if we aim for real-time model-checking in a live MAM. Furthermore, it is not clear that given the form of rules in this application, analysis of the rules requires a temporal logic. To a great extent, in this application, one could argue that the state of a sensor encapsulates a set of computational paths (or at least what is required to know them) and so we do not need to study the paths themselves. So, a SAT model is appropriate for this type of verification. Furthermore,

the verification then became so efficient it could be applied directly within the MAM, running in real time on the same computational hardware.

While we have only investigated one of the properties we mentioned Section 5, redundancy, how we detect and resolve redundancy depends also on our understanding of modalities, priorities, and more generally, context. For example, a user may not care about overlapping earcons *unless* one of them has been generated by a certain condition. For example, delivering messages via the television and the beeper simultaneously may be acceptable, unless one of the messages is considered significantly more urgent than the other, or has arisen because of an unsafe context. The area of semantics and ontologies for modalities/context requires further investigation. Acceptability and usability of modalities may be regarded as an example of crossing the "semantic rubicon"[5] [KA02]. A contribution of our formal modelling and analysis to MAM design has been to expose this crossing.

## 9 Related Work

Much formal analysis of pervasive systems is focussed on techniques for requirements involving location and resources, within a waterfall framework. For example, [CD09] employs the Ambient calculus for requirements and [CE07] employs a constraint-based modelling style and temporal logic properties. Some work has been done on better integration of formal analysis techniques within the context of interactive system interfaces (e.g. [CH08]), but there is little work on more tightly coupled models and analysis. One exception is [RBCB08], where a model of salience and cognitive load is developed and a usability property is considered. The model is expressed in a higher order logic, and the property is expressed in LTL. While our paradigm is different, the authors recognise they are engaged in a cyclic process. In some cases, their verification revealed inconsistencies between experimental behaviour and the formal model, which led them to suggest refinements to the rules and also new studies of behaviour. Finally, there is ongoing work to use policy conflict handling mechanisms embedded in telecommunications systems in homecare applications [WT08]. We believe our approach can provide a more generic framework for such a conflict management service.

## 10 Conclusions and future work

We have considered the problem of verifying context-aware, pervasive, interactive systems. These kinds of systems present numerous challenges for verification: context-changes may be difficult to model and predict and in addition, such systems often operate in situations that require practically unpredictable changes to the application functionality itself. We have outlined an approach to verification that makes explicit two different types of interaction: system configuration and system use. Our long term goal is to more tightly couple reasoning about configurable systems by configuring models, and closing the loop between design, use, configuration, modelling and verification. In particular, we are concerned with feeding back results of verification to users, designers, configurers, and modellers.

[5] the division between system and user for high level decision-making or physical-world semantics processing.

This paper reports on preliminary results from an example concerning an activity monitor from the MATCH homecare system. We have developed an event based general model in Promela, formulated and checked a number of properties in the model checker SPIN. We have concentrated on supporting end user configuration by checking for rule redundancy. Results from the general model led us to develop a specialised model for use with a SAT solver, and using that model we were able to verify an example set (taken from an actual log file), on the actual MAM, in real time.

The case study illustrates a number of engineering and foundational challenges for our approach: we have not modelled an idealised system, but one that has been designed and engineered in the context of specific practices and personal conventions. This presents non-trivial challenges for any modelling process. The work is still preliminary, but our results demonstrate proof of concept. A distinctive feature of the work is we generate automatically parts of the model from actual log files.

Longer term, our plans for further future work include generating more parts of models automatically from log files, for a class of context aware systems, and incorporating aspects of stochastic behaviour, performance, and real-time in the model and properties. We also plan to further investigate semantic models of modalities and context and the best way to present and use verification results, expecially in the context of human and non-human agents.

**Acknowledgements:**

This research is supported by the *VPS* project (*Verifying interoperability in pervasive systems)*, funded by the Engineering and Science Research Council (EPSRC) under grant number EP/F033206/1. We also acknowledge support from the MATCH Project, funded by the Scottish Funding Council under grant HR04016.

## Bibliography

[CD09] A. Coronato, G. De Pietro. Formal specification of a safety critical pervasive application for a nuclear medicine department. *International Conference on Advanced Information Networking and Applications Workshops*, pp. 1043–1048, 2009. doi:10.1109.WAINA.2009.198

[CE07] A. Cerone, N. Elbegbayan. Model-checking Driven Design of Interactive Systems. *Electron. Notes Theor. Comput. Sci.* 183:3–20, 2007. doi:dx.doi.org/10.1016/j.entcs.2007.01.058

[CH97] J. Campos, M. D. Harrison. Formally verifying interactive systems: a review. In *Design, Specification and Verification of Interactive Systems 97*. Pp. 109–124. Springer, 1997.

[CH08] J. C. Campos, M. D. Harrison. Systematic analysis of control panel interfaces using formal tools. In *XVth International Workshop on the Design, Verification and Specification of Interactive Systems (DSV-IS 2008)*. Lecture Notes in Computer Science 5136, pp. 72–85. Springer-Verlag, July 2008.

[CRB07] P. Curzon, R. Rŭkėnas, A. Blandford. An approach to formal verification of human-computer interaction. *Formal Aspects of Computing*, pp. 513–550, 2007. doi:10.10.1007/s00165-007-0035-6

[ES03] N. Eén, N. Sörensson. An Extensible SAT-solver. In Giunchiglia and Tacchella (eds.), *SAT*. Volume 2919, pp. 502–518. Springer, 2003.

[Hol03] G. J. Holzmann. *The SPIN model checker: primer and reference manual*. Addison Wesley, Boston, 2003.

[HS99] G. Holzmann, M. H. Smith. Software model checking - Extracting verification models from source code. In *Proc. FORTE/PSTV '99*. Pp. 481–497. Kluwer, 1999.

[JAK] Jake Project. http://code.google.com/p/jake-drivers/

[KA02] T. Kindberg, F. A. System software for ubiquituous computing. *Pervasive computing*, pp. 70–81, 2002.

[MG09] T. McBryan, P. Gray. User Configuration of Activity Awareness. *Lecture Notes In Computer Science* 5518:748–751, 2009. doi:dx.doi.org/10.1007/978-3-642-02481-8_113

[MRM09] P. Markopoulos, B. de Ruyter, W. E. Mackay. *Awareness Systems: Advances in Theory, Methology and Design*. Springer, 2009. doi:10.1007/978-1-84882-477-5

[RBCB08] R. Rŭkėnas, J. Back, P. Curzon, A. Blandford. Formal modelling of salience and cognitive load. *ENTCS*, pp. 57–75, 2008. doi:10.10.1016/j.entcs.2008.03.107

[RM07] Y. Riche, W. Mackay. MarkerClock: A Communicating Augmented Clock for the Elderly. *Proc. Interact 07. Part II, Lecture Notes In Computer Science* 4663:408–411, 2007.

[SHA] Shake users group. http://www.dcs.gla.ac.uk/research/shake/

[WMH07] J. Williamson, R. Murray-Smith, S. Hughes. Shoogle: excitatory multimodal interaction on mobile devices. *Proc. SIGCHI Conference on Human Factors in Computing Systems* 4663:121–124, 2007. doi:http://doi.acm.org/10.1145/1240624.1240642

[WT08] F. Wang, K. Turner. Policy Conflicts in Home Care Systems. *Proc. 9th Int. Conf. on Feature Interactions in Software and Communications Systems*, pp. 54–65, 2008.

# Markov Abstractions for Probabilistic $\pi$-Calculus

**Hugh Anderson**[1] **and Gabriel Ciobanu**[2]

[1] hugh@comp.nus.edu.sg
Wellington Institute of Technology,
New Zealand
[2] gabriel@iit.tuiasi.ro
Romanian Academy, Institute of Computer Science, Iaşi
and A.I.Cuza University of Iaşi, Romania.

**Abstract:** This paper presents a range of approaches to the analysis and development of program specifications that have been expressed in a probabilistic process algebra. The approach explores Markovian processes as a high-level abstraction tool to reason about system specifications. The abstractions include ones to check the structure of specifications, analyze the long-term stability of the system, and provide guidance to improve the specifications if they are found to be unstable. The approach could present interest to the formal methods and critical-systems development community, as it leads to an automatic analysis of some subtle properties of complex systems. We illustrate some aspects by analyzing the Monty Hall game, and a probabilistic protocol.

**Keywords:** Abstraction, probabilistic $\pi$-calculus, Markov processes.

## 1 Introduction

The process of effective abstraction underlies most facets of software production and analysis. When reasoning about software systems, we abstract out areas of interest, and reason only about those. When writing software, we use class declarations to encapsulate abstract notions. The benefit is that it could be easier to manipulate abstract notions. We can then make use of apriori knowledge about the abstraction. In the case of automatic analysis or automatic software production, we may also benefit from higher-level abstractions, the automatic tool using pre-proved transformations.

The approach presented here builds on earlier work in [11], where Markov Decision Processes (MDPs) are used as an abstraction in the context of the quantitative analysis of program predicate transformers, and in [12], where automated verification for probabilistic $\pi$-calculus is outlined.

We show the development of high-level abstractions based on Markovian processes. In the context presented here, the abstractions are used to assist in the analysis of specifications given in the probabilistic process algebra, useful for the analysis and development of probabilistic algorithms, and also the analysis of critical-systems in which we include estimates of the likelihood of failure. Such systems are commonly found in modern distributed computer systems, and a feature of this approach is that it supports mechanical formal analysis of the systems.

The approach starts with the generation of a Markov transition matrix for the program specification. After generating the transition matrix, we can reason directly about the matrix, or

establish the long-term behaviour/equilibrium state of the system, or analyze it directly by using the eigenvectors of the matrix. In each case, we use known properties of the matrix to pick out an area of the specification to fix/modify.

In order to follow this technical sequence, we need to provide effective abstractions of the elements of the approach. Section 2 provides background material on Markovian processes and transition matrices, as well as the probabilistic $\pi$-calculus. Section 3 continues with a discussion on suitable abstractions. Section 4 shows various worked examples, including the analysis of a probabilistic protocol. Section 5 concludes the paper.

## 2 Preliminaries

In this section, the underlying components of the approach, probabilistic process algebras and Markov chains, are briefly introduced and defined.

A process algebra is a technique for mathematically modeling systems constructed of interacting concurrent processes. The technique deserves the term algebra, as it is concerned with axioms and algebraic transformations of expressions in the process algebra.

The most well-known process algebras are the Communicating Sequential Processes (CSP), the Calculus of Communicating Systems (CCS) and the $\pi$-calculus. CSP is presented in [1]; CCS and the $\pi$-calculus are presented in [10]. Each of these process algebras is useful in its own right, concerned mostly with notions of equivalence between expressions in the process algebras, or with notions of satisfaction between a process algebra expression, and some property expressed in a modal logic.

Probabilistic process algebras have been used for modelling complex systems. Such systems may include elements of interaction where the environment introduces uncertainty; for example, the behaviour of people interacting with the processes, or communication bit error rates, or speed.

The advantage of model description in process algebra is compositionality, i.e., that complex models can be built from smaller ones. With probabilistic process algebra, it is generally understood that the underlying quantitative semantics of the concurrent model of computation is a discrete or continuous-time Markov chain [8]. The analysis of continuous-time Markov chains with large populations can be computationally quite expensive, and so deterministic models can be more efficient.

### 2.1 Probabilistic $\pi$-calculus

Probabilistic $\pi$-calculus is an extension of the $\pi$-calculus introduced in [12] with the aim of modeling performances of dynamically reconfigurable systems. It inherits all the syntax of $\pi$-calculus, and extends it with the possibility of associating to each action a probability distribution. This means that it is possible to associate to each prefix a quantitative value, represented by the value of a random variable, which follows the above mentioned probability distribution. Distributed systems often have probabilities associated with them that can be represented in the probabilistic $\pi$-calculus.

Many works have pointed out the usefulness of probabilistic and stochastic versions of the $\pi$-calculus in modeling various systems (see, e.g., [13, 2, 3]). Mainly it could be applied to

labeled transition systems representing concurrent systems; this is not possible by using differential equation models. For instance, the use of process algebras in systems biology is natural and provides double advantages: on the one hand, the representation is incremental and compositional; on the other hand, the models support formal verification techniques such as behavioral equivalences and model checking [4, 5, 12].

**Definition 1 (Probabilistic $\pi$-calculus)** Let $\mathcal{N} = \{a, b, \ldots, x, y, \ldots\}$ be an infinite set of names. A probabilistic $\pi$-calculus process is an expression using the following grammar:

$$P, Q ::= 0 \mid \sum_i \langle \pi_i, r_i \rangle .P_i \mid (\nu x)P \mid [x = y]P \mid P|Q \mid P(y_1, \ldots, y_n)$$

where $\pi$ is either $x(y)$ or $\bar{x}y$ or $\tau$, and $r \in \mathbb{R}^+ \cup \{\infty\}$.

The intuitive meaning of the operators is essentially the same as in $\pi$-calculus. $x(y)$ denotes that we are waiting for a message on the channel $x$ and $y$ acts as a placeholder, which will be replaced with the received message. $\bar{x}y$ represents the output of the message $y$ on the channel $x$. $\tau$ is the silent action. Standard considerations about free and bound names hold. In general, inputs are binding operators on the arguments. This means that in the process $x(y).P$ the name $y$ is bound in $P$, and not accessible from outside $P$. Restriction $(\nu n)P$ of the name $n$ makes that name private and unique to $P$: the name $n$ becomes bound in $P$. Recursion models infinite behaviour by assuming the existence of a set of equations of the form $A(x) = P$ such that $x \in fn(P)$, where $fn(P)$ stands for the usual free names of $P$. The definition of $fn(P)$ is standard taking into account that the only binding operators are inputs and restriction.

0 represents the inactive process. The probabilistic process $\langle \pi, r \rangle .P$ is used in a probabilistic choice operation, so the process $\langle \tau, 0.5 \rangle .P + \langle \tau, 0.5 \rangle .Q$ will either continue with process $P$ or $Q$, with equal probability. If a probability $r$ is 1.0, then we may omit it. In this probabilistic $\pi$-calculus, we do not consider nondeterminism.

## 2.2 Markovian Processes

There are many different kinds of processes, however a particular subset of all processes, the Markov processes, have been studied in great detail for many years.

Markov processes are viewed as a set of random variables $\{X_t\}$, where the time index $t$ runs through an ordered set. The set of all possible values of the variables is known as the *state space* of the process. For one-dimensional state spaces, we classify Markov processes into four distinct categories:

| | **Time** | **State space** |
|---|---|---|
| 1 | discrete | discrete |
| 2 | continuous | discrete |
| 3 | discrete | continuous |
| 4 | continuous | continuous |

A Markov chain is an abstraction $\mathcal{M}$ representing a probabilistic process in terms of a set of states $\mathcal{S}$, and a probability transition matrix $\mathcal{T}$ from one set of states to another. Transitions

from state to state occur at discrete time intervals. The future behaviour of a Markov chain is not dependent on the previous path arriving at the current state, although we can specify an initial state according to some pre-defined probability over $\mathscr{S}$.

In elementary probability theory, given an event $B$, and an exhaustive set of mutually exclusive events $\{C_i\}$, then $\text{prob}(B) = \sum_i \text{prob}(B \mid C_i)\,\text{prob}(C_i)$. In a continuous state space, the conditional probability density function at time $n$ given the state occupied at time $m$ ($l < m < n$) is:

$$p_{X_n}(x \mid X_l = z) = \int_{-\infty}^{\infty} p_{X_m}(y \mid X_l = z)\, p_{X_n}(x \mid X_m = y, X_l = z)\, dy \tag{1}$$

where the $X_l$ are random variables specifying the states of a probabilistic process. A probabilistic process is a Markov Process if, for arbitrary times $l < m < n$,

$$p_{X_n}(x \mid X_m = y, X_l = z, \ldots) = p_{X_n}(x \mid X_m = y) \tag{2}$$

That is, probability density functions are only dependant on the most recent of the time points. Given equation (2), we can rewrite equation (1) as:

$$p_{X_n}(x \mid X_l = z) = \int_{-\infty}^{\infty} p_{X_m}(y \mid X_l = z)\, p_{X_n}(x \mid X_m = y)\, dy \tag{3}$$

This is called the Chapman-Kolmogorov equation, and it indicates that it is possible to build up probability density functions over a long period of time $(l \ldots n)$ from short time periods $(l \ldots m)$ and $(m \ldots n)$. We can express this for discrete states as:

$$p^n_{jk} = \sum_{i=0}^{\infty} p^{n-1}_{ji}\, p_{ik}$$

and note that in process algebras we are dealing with discrete state spaces and hence Markov *chains*, rather than the continuous state space and hence Markov *processes*.

A Markov Decision Process extends the notion of the Discrete Time Markov chain (DTMC) by allowing the process to be controlled through *actions* at each state. MDPs also provide the notion of reward for each taken action and each pair of present and subsequent states. By contrast, in the case of DTMCs, the process cannot be controlled. The subsequent states are only chosen in a probabilistic fashion, with respect to some prescribed transition matrix. Thus DTMCs only allow probabilistic choice (through the transition matrix), while MDPs allow both probabilistic choice (through the probability function) and nondeterministic choice (through the set of possible actions from each state).

If a system eventually settles to a state of statistical equilibrium represented by a state distribution vector $\pi = (\pi_1, \ldots, \pi_n)$, where $\pi_1 + \ldots + \pi_n = 1$, then $\pi = \pi\mathscr{T}$ or alternatively $\pi(\mathscr{I} - \mathscr{T}) = 0$. This gives us a homogeneous set of equations, which will have a solution if $\mathscr{I} - \mathscr{T}$ vanishes. There are various standard methods for solving such a set of equations.

Another notion is that of class solidarity: a state of a given type can only intercommunicate with states of the same type. Furthermore they must have the same recurrence period. A set of states that communicate only with each other are called a closed set - an absorbing set of

states. The decomposition theorem states that we may divide any Markov chain into two sets - the recurrent and the transient states. The recurrent set may be decomposed into closed sets. Within each closed set, all states communicate with same period.

The Perron-Frobenius theorem asserts that if $A$ is of order $h \times h$ and if $A$ has $t > 1$ eigenvalues equal in modulus to $\lambda_1$ (the largest one), then $A$ can be reduced to a cyclic form by a permutation applied to both rows and columns. The import of this is that it allows us to differentiate between cyclic and stochastic processes.

Hansson and Jonsson have developed PCTL in [7]. It is a probabilistic real-time computation tree logic for checking discrete time Markov chains. PCTL path and state formulas represent properties of states and sequences of states. The PCTL formula are applied to discrete time Markov chains, yielding judgements over them that indicate if the chain *satisfies* the formula. This model-checking approach is not considered here.

## 3 Markov Abstractions

The principal abstraction is just the (perhaps obvious) one that the transition matrix derived from the specification is an abstraction of the specification. Observations about the matrix apply to the specification. We begin with some general observations about the analysis of transition matrices:

1. Given a Markov transition matrix $\mathscr{T}$, the solution of the equation $p\mathscr{T} = p$ can give $p$, the steady-state (or equilibrium) state value matrix[1].

2. The least or greatest expected returns from a state $p_n$, can be easily calculated using the transition matrix $\mathscr{T}$. The calculations are easy to do, but the *meaning* associated with them needs to be given an intuition. This intuition is clarified by means of examples: "Given that we are at $p_n$ what is the least expected cost of a decision taken here?"

3. More complex distributions are possible. For example - the $t$ value may itself be a MDP, making the decision based on (say) a dial on the black box. This suggests a *testing* process - given a specification, and a particular MDP suggesting how another process/tester will interact with it.

We also have the following abstractions over the transition matrix, which can be tied back to the particular state(s) that give rise to the effect. This gives us a method for specification improvement, involving testing the transition matrix for each effect, and then relating this back to the causative states.

Given a Markov transition matrix $\mathscr{T}$,

1. determine which (if any) states are *absorbing* (i.e. capturing states of the system).

2. determine which (if any) states are *ephemeral* (i.e. unreachable).

3. determine if the system is *cyclic* or *ergodic*. Is it expected to be cyclic? Is it expected to be a distribution?

[1] There are two possible kinds of outcomes - cyclic ones, and steady state ones.

4. if it is cyclic, determine the *states* of the cyclic behaviour.
5. determine if the system is a single Markov chain or two (or more) Markov chains.
6. if it describes two (or more) Markov chains, determine the *states* belonging to each of the chains.
7. the distribution of *eigenvalues* of $\mathscr{T}$ gives the following insights:
   (a) If one is 0, then the transitions are not invertible
   (b) If two are close together, then it may take a long time to stabilize
   (c) The maximum value gives the long term behaviour
   (d) The case when the behaviour is cyclic (i.e. no equilibrium)

For each abstraction, we develop an intuition or interpretation, and show how it can be mechanically calculated using conventional linear programming tricks.

### 3.1 Markov Classification of States

The states of a Markov chain are classified as follows:

| **Type of state** | **Definition** |
|---|---|
| Ephemeral | Cannot be reached from any other state |
| Absorbing | Cannot ever leave this state |
| Periodic | Return to this state cyclically |
| Aperiodic | Not periodic |
| Recurrent | Eventual return certain |
| Transient | Eventual return uncertain |
| Positive-recurrent | Recurrent, finite mean recurrence time |
| Null-recurrent | Recurrent, infinite mean recurrence time |
| Ergodic | Aperiodic, positive recurrent |

Each of these types of states has some relevance to the analysis of probabilistic process algebra. For example an ephemeral state (if it is not the first state in the system), may indicate an unreachable part of a specification. An absorbing state may indicate a deadlock condition.

## 4 Direct Analysis of Systems

In this section, some simple examples are used to demonstrate how systems expressed in a probabilistic process algebra may be expressed as transition matrices, which are then examined directly in terms of the Markov classification of states, in order to discover various properties.

In each example, we also give a motivation or intuition about the applicability of the particular technique used.

### 4.1 Periodic versus Ergodic

It is useful to discover if an arbitrary process algebra expression is periodic or ergodic. This may not be apparent from the process algebra expression, and if it is found to be (say) periodic when we expect it to be random, then this may indicate a problem with the expression. Let us begin with a very simple example:

$$\begin{aligned} P &\stackrel{def}{=} \langle a, 0.5\rangle .P + \langle b, 0.5\rangle .Q \\ Q &\stackrel{def}{=} \langle a, 0.5\rangle .P + \langle b, 0.5\rangle .Q \end{aligned}$$

The prefixes $\langle x, r\rangle$ indicate a pair consisting of the prefix, and its probability. This one's transition matrix is $\begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix}$ with an equilibrium state vector of $(0.5, 0.5)$. Consider also the expression:

$$\begin{aligned} P &= \langle a, 0\rangle .P + \langle b, 1.0\rangle .Q \\ Q &= \langle a, 1.0\rangle .P + \langle b, 0\rangle .Q \end{aligned}$$

with the transition matrix $\begin{bmatrix} 0 & 1.0 \\ 1.0 & 0 \end{bmatrix}$ and, again, an equilibrium state vector of $(0.5, 0.5)$.

Can we differentiate between these two? They *are* quite different, the first representing a probabilistic process, producing strings of $a$'s and $b$'s, the second representing a deterministic process producing an alternating string of $a$'s and $b$'s. If we look at the eigenvalues of the two matrices, we see that the first matrix returns the eigenvalues 1 and 0, whereas the second produces 1 and $-1$. From the Perron-Frobenius theorem, we can immediately tell from the two eigenvalues that this matrix represents a cyclic process. In summary:

> *The direct analysis of the eigenvalues of a transition matrix allows us to differentiate between two different types of process - specifically cyclic and stochastic processes.*

This analysis may be useful in either case - when we expect a stochastic process and get a cyclic one, and *vice-versa*.

### 4.2 Ephemeral States

The trivial ephemeral state $S_i$ in a Markov chain is a state in which the $i$-th column of the transition matrix is *all-zeroes*. This indicates that you can only pass out of this state, and never get back into it.

The intuition in process algebra terms is that we might consider these states to be correct if they are initial states, but otherwise they are *dead-code*. You can never get to an ephemeral state if you do not start in the ephemeral state. In this example, column 4 is all-zeroes, indicating that

$S_4$ is ephemeral:

$$P = \begin{bmatrix} 0.3 & 0.0 & 0.0 & 0.0 & 0.0 & 0.7 \\ 0.0 & 0.1 & 0.1 & 0.0 & 0.0 & 0.8 \\ 0.0 & 0.0 & 0.8 & 0.0 & 0.0 & 0.2 \\ 0.2 & 0.0 & 0.0 & 0.0 & 0.4 & 0.4 \\ 0.0 & 0.5 & 0.0 & 0.0 & 0.0 & 0.5 \\ 0.1 & 0.3 & 0.2 & 0.0 & 0.0 & 0.4 \end{bmatrix}$$

The concept of the ephemeral state may be extended to an ephemeral state-set. This is a set of states which you can only pass out of. In this case, there can be no periodic structure in the ephemeral state-set, and one of the states must therefore be a first ephemeral state. If it is removed from the transition matrix, we are left with a transition matrix with a (possibly null) ephemeral state-set, and the rest of the matrix. In this way, we can step-wise remove the ephemeral state set from the transition matrix, identifying all its elements as we go.

$$P' = \begin{bmatrix} 0.3 & 0.0 & 0.0 & 0.0 & 0.7 \\ 0.0 & 0.1 & 0.1 & 0.0 & 0.8 \\ 0.0 & 0.0 & 0.8 & 0.0 & 0.2 \\ 0.0 & 0.5 & 0.0 & 0.0 & 0.5 \\ 0.1 & 0.3 & 0.2 & 0.0 & 0.4 \end{bmatrix} \qquad P'' = \begin{bmatrix} 0.3 & 0.0 & 0.0 & 0.7 \\ 0.0 & 0.1 & 0.1 & 0.8 \\ 0.0 & 0.0 & 0.8 & 0.2 \\ 0.1 & 0.3 & 0.2 & 0.4 \end{bmatrix}$$

In this example, by removing $S_4$ we discovered a second ephemeral state $S'_4$. Removing this state removes the ephemeral state-set entirely. In summary:

> *The ephemeral states should correspond only to initial states. If there are any ephemeral states that are not the initial ones, then these indicate that the states are dead states, i.e. ones that will never be visited.*

This analysis is useful in finding parts of a specification that are not required. If so they may as well be removed.

### 4.3 Absorbing States and State Sets

An absorbing state $S_i$ in a Markov chain is one in which the $i$-th row is *all-zeroes*. This indicates that you can only pass into this state, and never get back out of it. The intuition in process algebra terms is that we might consider these states to be deadlock ones - our process has arrived at this state and can never exit it. The concept of the absorbing state may be extended to an absorbing state-set, which may either be a chain of states ending in an absorbing state, or a set of states which may be either ergodic or periodic, but in any case do not ever exit.

The first option is easily handled in a manner analogous to that used in the presentation for ephemeral states; the second is a little more difficult. However, the Markov techniques provide again a solution. By permuting the rows and columns, we transform the matrix until it is $P = \begin{bmatrix} \cdots & \cdots \\ 0 & [K] \end{bmatrix}$. In this matrix, the bottom right hand corner $K$ is a square matrix, with all-zero entries to the left of it. If the states contributing to the rows of $K$ do not contain an initial state, we can deduce that this is a set of states that are absorbing. In summary:

> *The absorbing states should correspond only to final states. If there are any absorbing states or state sets that are not final, then these indicate that the states are deadlock states.*

This analysis is useful in finding parts of a specification that cause deadlocks. If so, these are better removed early (at the specification stage) than later.

## 4.4 Direct Computation

It is relatively easy to manipulate the matrices directly to discover properties of a process. This is illustrated by examining a process algebra expression intended to capture the elements of the Monty-Hall game. The rules of the Monty-Hall game are as follows:

> *A contestant appears in a TV show. The announcer for the show (Monty Hall) hides a prize in an alcove behind one of three curtains. The contestant is asked to select one of the curtains, and then announce the selection. Since the announcer for the show knows where the prize is located, he opens one of the other two curtains, showing the contestant that the prize is not behind it. The contestant is then asked if she wants to change her mind - she can either stick to the original curtain, or change to the other curtain.*

The question is: which of these options should the contestant choose? The surprising answer is that she should change her mind.

### 4.4.1 Parallel Formulation

We begin with a probabilistic $\pi$-calculus expression of the game, played with two interacting processes representing Monty Hall (monty), a contestant (contestant). Beginning with the Monty Hall process, we have Monty selecting a random prize curtain, and signalling the contestant on channel $x$. The contestant replies with a curtain selection on channel $z$. Monty then signals the contestant on channel $y$ with a revealed curtain. Finally the contestant replies with a choice on channel $w$. Monty then signals either the win or lose on channel public:

$$
\begin{array}{lcl}
\text{game} & \stackrel{\text{def}}{=} & \text{monty} \mid \text{contestant} \\
\text{monty} & \stackrel{\text{def}}{=} & \sum_{i\in\{1,2,3\}} \left\langle \bar{x}, \frac{1}{3} \right\rangle .z(c).\text{select}_{ic} \\
\text{select}_{ic} & \stackrel{\text{def}}{=} & \begin{cases} \left\langle \bar{y}(c_l), \frac{1}{2} \right\rangle .w(f).\text{signal}_{if} + \left\langle \bar{y}(c_r), \frac{1}{2} \right\rangle .w(f).\text{signal}_{if} & \text{if } i = c \\ \bar{y}(c_o).w(f).\text{signal}_{if} & \text{if } i \neq c \end{cases} \\
\text{signal}_{if} & \stackrel{\text{def}}{=} & \begin{cases} \overline{\text{public}}(\text{win}).!\text{monty} & \text{if } i = f \quad (\text{a win!}) \\ \overline{\text{public}}(\text{lose}).!\text{monty} & \text{if } i \neq f \quad (\text{a lose}) \end{cases}
\end{array}
$$

The process is replicated, and continues forever, or until we switch off the TV. The other processes may be defined as follows, where the contestant makes the deliberate decision to choose the *other* curtain (that is the contestant changes curtains):

$$
\text{contestant} \stackrel{\text{def}}{=} x.\left( \sum_{j\in\{1,2,3\}} \left\langle \bar{z}(c_j), \frac{1}{3} \right\rangle .y(m).\bar{w}(c_o).!\text{contestant} \right)
$$

To model the the case when the contestant decides not to change curtains, our contestant process would be

$$\text{contestant} \stackrel{\text{def}}{=} x.\left(\sum_{j\in\{1,2,3\}} \left\langle \bar{z}(c_j), \frac{1}{3}\right\rangle .y(m).\bar{w}(c_j).!\text{contestant}\right)$$

We can model the whole system using a simple (single) process algebra expression, only consisting of choice, and no parallel composition. The following section shows such an expression.

### 4.4.2 Expansion of Parallel Composition - 1

We can expand the parallel composition of the expression using the modified expansion law as shown below. We assume that we have actions $(a_1, a_2, a_3)$ representing the selection of Monty Hall selecting one of the three alcoves. The contestant then can indicate with one of three actions $(b_1, b_2, b_3)$ which alcove/curtain she wishes to choose by pressing the corresponding button. Monty can then open one of the two remaining curtains, the left one or the right one if there is a choice $(c_l, c_r)$, or the only remaining one if there is not $(c_o)$. Finally, the contestant can choose either to change to the other curtain by selecting action $(d)$ or not. In the following expression, the contestant tries the *changing* algorithm - that is, she changes her selection. There are also two extra states, indicating *winning* and *losing* which we are interested in:

$$\begin{aligned}
\text{hide} &\stackrel{\text{def}}{=} \sum_{i\in\{1,2,3\}} \left\langle a_i, \frac{1}{3}\right\rangle .\text{select}_i \\
\text{select}_i &\stackrel{\text{def}}{=} \sum_{j\in\{1,2,3\}} \left\langle b_j, \frac{1}{3}\right\rangle .\text{monty}_{ij} \\
\text{monty}_{ij} &\stackrel{\text{def}}{=} \begin{cases} \langle c_l, 0.5\rangle .\text{change}_l + \langle c_r, 0.5\rangle .\text{change}_r & \text{if } i = j \\ c_0.\text{change}_o & \text{if } i \neq j \wedge k \neq i \wedge k \neq j \end{cases} \\
\text{change}_i &\stackrel{\text{def}}{=} \begin{cases} d.\text{lose} & \text{if } i = l \\ d.\text{lose} & \text{if } i = r \\ d.\text{win} & \text{if } i = o \end{cases} \\
\text{lose} &\stackrel{\text{def}}{=} \text{hide} \\
\text{win} &\stackrel{\text{def}}{=} \text{hide}
\end{aligned}$$

We can now use this to generate a simple Markov transition matrix, as seen in section 4.4.4.

### 4.4.3 Expansion of Parallel Composition - 2

Another technique may be used to expand out the parallel composition of probabilistic processes. In [9], Hillston et al show how the Kronecker representation of a parallel composition of Markovian processes is an efficient representation, and allows us to create the generator matrix as required. The Kronecker representation of a parallel composition of $N$ component processes is represented by a computation over their transition matrices $R_i$, and including factors related to their *interaction* ($P_{i,\alpha}$, which represents a probability transition matrix for each component $i$

associated with each interaction $\alpha$), and *normalization* ($\overline{P}_{i,\alpha}$, which normalizes the interaction matrices for each component $i$ and each interaction $\alpha$, where $r_\alpha$ is the minimum of the rates of action $\alpha$):

$$Q \stackrel{\text{def}}{=} \bigoplus_{i=1}^{N} R_i + \sum_{\alpha \in Z} r_\alpha \left( \bigotimes_{i=1}^{N} P_{i,\alpha} - \bigotimes_{i=1}^{N} \overline{P}_{i,\alpha} \right)$$

In our example, if the matrix for monty was $\mathscr{M}$ and the matrix for the contestant was $\mathscr{C}$, then the Kronecker representation is $Q = \mathscr{M} \oplus \mathscr{C} + f \times [\mathscr{M}_i \otimes \mathscr{C}_i - \mathscr{M}_n \otimes \mathscr{C}_n]$. Note that this representation is compositional, and this is of use in that the sort of sparse matrices generated by our process algebra expressions are *compactly* represented by the Kronecker expression. In addition, the unexpanded expression may *itself* give insights into the behaviour of the interacting processes - for example, the interaction matrix may tell you about the *degree of binding* between processes.

### 4.4.4 Markov Transition System

In our example, the first process expansion technique applied to the process expression results in 18 states - i.e. the size of the transition matrix is $18 \times 18$. The matrix is:

$$M = \begin{array}{r} \\ \text{hide} \\ \text{select}_1 \\ \text{select}_2 \\ \text{select}_3 \\ \text{monty}_{11} \\ \text{monty}_{12} \\ \text{monty}_{13} \\ \text{monty}_{21} \\ \text{monty}_{22} \\ \text{monty}_{23} \\ \text{monty}_{31} \\ \text{monty}_{32} \\ \text{monty}_{33} \\ \text{change}_l \\ \text{change}_r \\ \text{change}_o \\ \text{lose} \\ \text{win} \end{array} \begin{bmatrix} 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Noting that the process cycles after four states, we calculate game $=$ init $\times M^4$ where init is the initial state vector. From this we get a vector representing the probability of being in each state after four cycles. The vector is game $= (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,\frac{1}{3},\frac{2}{3})$. The values associated with lose and win are $\frac{1}{3}$ and $\frac{2}{3}$ respectively, indicating that with this strategy, we will

(in the long term) win! By contrast, if we change the definition of the $\text{change}_{ij}$ state to represent the choice of not changing curtains:

$$\text{change}_{ij} \stackrel{\text{def}}{=} \begin{cases} \text{win} & \text{if } i = l \\ \text{win} & \text{if } i = r \\ \text{lose} & \text{if } i = o \end{cases}$$

By examination of this new matrix, we establish that the values associated with lose and win are $\frac{2}{3}$ and $\frac{1}{3}$ respectively, indicating that this is not as good a strategy as the previous one.

### 4.4.5 Comment on Direct Evaluation

The transition laws for $S\pi$ may be directly applied to the original expression of the game, but they do not result in an efficient reduction of the expression. Initially there are three possible transitions, and each must be separately evaluated - i.e. the expression expands rather than reduces. The point here is that the matrix representation is simpler to handle, and reduces mechanically and quickly. In this section various techniques for evaluating the properties of probabilistic processes were given. In summary:

> *The reduction of a parallel composition of probabilistic processes to a single Markov transition matrix or chain allows a compact representation of the behaviour of the process. This matrix/chain may then be evaluated using either algebraic or arithmetic techniques to discover properties of the original processes.*

This analysis is useful in directly evaluating the probabilistic behaviour of a process without having to apply the more complex $S\pi$ rules a step at a time. The evaluation is done in a simple manner on the transition matrix alone.

## 4.5 First Passage Probabilities

In a previous example, we were interested in the time taken to get to the absorbing state set. In Markov terms, this is known as the first passage probability. The first passage probability from state $i$ to state $j$ at time $t$ is defined by the conditional probability that state $j$ is entered at time $t$, and state $j$ is not entered *before* time $t$, this being conditional on starting at state $i$. The mean first passage probability $M_{ij}$ from state $i$ to state $j$ in transition matrix $\mathscr{T}$ is derived from the mean first passage matrix, which is given by

$$M = (I - Z + EZ_{\text{diag}})D$$

where $I$ is the identity matrix, $Z$ is the fundamental matrix, $E$ is a matrix containing ones everywhere, $Z_{\text{diag}}$ is the matrix containing in its diagonal the components of the fundamental matrix (and zeros everywhere else) and finally, $D$ contains in its diagonal $1/\alpha_i$. (1 divided by the components of the limit matrix). $Z$, the fundamental matrix, is computed by $Z = (I - (P - \mathscr{T}))^{-1}$.

This gives us a technique for finding the mean passage time - the expected number of transitions/steps performed to get from state $i$ to state $j$. We begin by constructing the transition matrix for our probabilistic expression. We next ensure that it is irreducible and stochastic, and we may

then calculate the mean first passage probability matrix $M$. This matrix may be used to evaluate mean first passage times for any transition of interest. Note that once again, this computation is performed by manipulations of the transition matrix $\mathscr{T}$, which is used to derive the mean first passage matrix. This matrix may in turn be used to derive specific useful properties as seen in the next two sections.

### 4.5.1 Commute Time

A useful property that may be of interest is the *commute* time of a system. The commute time between state $i$ and state $j$ is the expected time to return to state $i$ after visiting state $j$ at least once. This is derived from the mean first passage matrix:

$$C_{ij} = M_{ij} + M_{ji}$$

An intuition about the usefulness of *commute* time may be gleaned from the following:

> *Consider the evaluation of two competing specifications for a probabilistic algorithm. A particular cycle of states in the algorithm is of interest, as it is time critical. The* commute*-time analysis will yield the faster specification.*

This quantitative evaluation of two competing designs may assist in the correct selection of a better design for implementation.

### 4.5.2 Cover Time

The *cover* time is the expected time to visit all components of a system. Again this has an interpretation in the software design, specification and analysis field. The calculation of the cover time is not as simple as the commute time, and involves analysis of the spanning tree for the transition graph. However tight bounds may be efficiently calculated as seen in [6]. An intuition about the usefulness of *cover* time may be gleaned from the following:

> *Consider the evaluation of two competing specifications for a probabilistic algorithm. Each of the states in the algorithm is of interest, and must be visited at least once, and this is time critical. The* cover*-time analysis may yield the better one of the two competing specifications.*

Again, quantitative evaluation of two competing designs helps in the correct selection of a better design for implementation.

## 4.6 Equilibrium

Another core concept in the analysis of ergodic Markov processes is that of equilibrium. The calculation of the equilibrium of a Markov chain gives insight into the long-term behaviour of the process. As an example of this, we consider an expression of a probabilistic algorithm for a non-repudiation protocol. The purpose of this protocol is to ensure that two processes agree that a transaction has taken place. At some time, one of the processes cannot violate the protocol and later claim that it did not. The protocol probabilistically ensures that neither process

can repudiate an agreement. We consider two expressions of the protocol, one in which both processes act honourably, and a second in which the second process attempts to cheat by refusing to send an acknowledgement. By looking at the long-term behaviour of the two expressions, using an infinite Markov chain, we see that in the first case the likelihood of agreement is unity. However in the second case the likelihood of agreement approaches zero.

This protocol is simplified for the purposes of the paper. The sender process randomly chooses a message $n$ out of $M$ possible message numbers, and elects to transmit the (encrypted) transaction during this message. A friendly receiver immediately responds with a acknowledgement containing the message, and then proceeds to decrypt the message (this process taking much longer than the round-trip time of the messages). When the message decrypts correctly, the protocol has ended, and the receiver cannot later attempt to deny the transaction, as it sent the acknowledgement before it attempted to decrypt. An unfriendly receiver flips a coin to decide if it wishes to NOT send an acknowledgement. If the process is lucky enough to do this when it receives the correctly encrypted transaction, then it can deny the transaction ever took place. We model the system with a sender and a receiver. The $x$ channel is used for the messages from the sender to the receiver, the $y$ channel contains the acknowledgement. Beginning with the sender process, and then defining a good and a bad receiver process, we have:

$$
\begin{array}{lcl}
\text{protocol} & \stackrel{\text{def}}{=} & \text{sender} \mid \text{receiver} \\
\text{sender} & \stackrel{\text{def}}{=} & \left\langle \bar{x}(t,d(t)), \frac{1}{M} \right\rangle .y.0 + \left\langle \bar{x}(q,d(r)), 1-\frac{1}{M} \right\rangle .y.!\text{sender} \\
\text{goodie} & \stackrel{\text{def}}{=} & \begin{cases} \langle x(t,q), 1.0 \rangle .\bar{y}.\overline{\text{public}}(T).0 & \text{if } d(q) = t \\ \langle x(t,q), 1.0 \rangle .\bar{y}.!\text{goodie} & \text{if } d(q) \neq t \end{cases} \\
\text{baddie} & \stackrel{\text{def}}{=} & \begin{cases} \left\langle x(t,q), \frac{1}{2} \right\rangle .\overline{\text{public}}(T).0 + \left\langle x(t,q), \frac{1}{2} \right\rangle .\bar{y}.!\text{baddie} & \text{if } d(q) = t \\ \left\langle x(t,q), \frac{1}{2} \right\rangle .\overline{\text{public}}(F).0 + \left\langle x(t,q), \frac{1}{2} \right\rangle .\bar{y}.!\text{baddie} & \text{if } d(q) \neq t \end{cases}
\end{array}
$$

Choosing an arbitrary (finite) value for $M$, it is easy to calculate that the probability for the *goodie* to succeed will be 1.0, and for the *baddie* reduces to zero:

$$P(T) = \frac{1}{M}\sum_{i=1}^{M} 1 \text{ if } d(q) = t \quad (=1) \quad \text{and} \quad P(T) = \frac{1}{M}\sum_{i=1}^{M} \left(\frac{1}{2}\right)^i \text{ if } d(q) = t \quad (\to 0)$$

In summary:

> *The equilibrium for a stochastic process is easily derived from the transition matrix, and allows us to make assertions about the long term behaviour of the process.*

The analysis is completely mechanical, and relies on an early use of abstraction, reducing a relatively complicated process algebra expression to manipulations on a Markov chain.

## 4.7 Evaluation of System Behaviour: a *hard* problem

The final approach to the analysis of communicating systems suggested by the Markov abstraction is the prediction and evaluation of system behaviour even when we cannot complete the quantification of the transition matrix. The view here is that we may not be able to quantify the probabilities of certain events, or, because of the interaction with other events, be unable to externally monitor them. In this situation, we can still generate a Markov transition matrix, although the matrix will contain variables (unknowns) in certain places.

Consider a tiny example of a transition matrix $\mathscr{T}$: $\mathscr{T} = \begin{bmatrix} p & 1-p \\ q & 1-q \end{bmatrix}$. To discover the eigenvalues of this, we need to solve $\det\left(\begin{bmatrix} p-\lambda & 1-p \\ q & 1-q-\lambda \end{bmatrix}\right) = 0$ for $\lambda$. This is simple for the given problem: $\lambda^2 + (q-p-1)\lambda + (p-q) = 0 \;\Rightarrow\; \lambda = 1, p-q$. Note that the single large value 1 indicates that this is a stochastic matrix if $p,q \notin \{0,1\}$. The solutions for $\lambda$ are then used to generate the eigenvectors for the matrix $\mathscr{T}$ by solving $v\mathscr{T} = \lambda v$ for $v$. However, in a more general case, this is normally considered a *hard* problem, particularly if there are a large number of variables and the matrix is large. Instead, this sort of problem is solved using arithmetic and algorithmic techniques rather than algebraic ones. Again there are many well-known arithmetic techniques for quickly finding solutions to large sets of equations. Let us now consider how this sort of evaluation can assist in a software engineering process:

> *Consider the evaluation of a specification for a probabilistic algorithm in which some of the probabilities are unknown. By constructing a transition matrix, and then solving the eigenvector equations, we derive a compact set of information that can be used to make assertions about the specification. (For example - in the trivial example above, if $p = 1$ and $q = 0$ we can immediately tell that the process is cyclic with length 2. If $p = q$ we can immediately tell that the process has an ephemeral state. For any other values for p and q, the process is stochastic).*

In this example, we see how the Markov abstraction can lead to a better understanding of a process even in the absence of quantitative values.

## 5 Conclusion

In this paper we have concentrated on the mechanical calculation of properties of probabilistic process algebra expressions using a matrix $P$ which defines a Markov decision process for the expression. The key point of the approach is that instead of reasoning about the detailed structure of the process algebra expression, we reason at a higher level of abstraction using Markovian abstractions. These abstractions include ones to predict long term behaviour of a system, identify deadlock states, identify ephemeral states, and calculate system properties by direct manipulation of $P$. Note that this approach is different from previous approaches, which do not concentrate on the element of abstraction suggested by the use of Markovian processes.

This abstraction allows us to compactly represent aspects of the behaviour of processes. For example, a single vector (the eigenvalues of the matrix) can be used to infer useful properties of a process. In addition, efficient libraries and procedures for calculating probabilities or rates have been developed over the long history of Markov processes, and we can obtain results using these efficient arithmetic techniques. The results can not only give direct quantitative assessments of the behaviour of a design or software element, but also can lead to comparisons between competing designs. If two designs/implementations had similar properties in other areas, but one was better in it's *cover* time, then we might choose to pick this one.

In summary, we have outlined ways in which Markovian processes may be used as a high-level abstraction tool to reason about program specifications expressed in a probabilistic $\pi$-calculus.

The abstractions include ones to check the structure of specifications, analyze the long-term stability of the system, and provide guidance to improve the specifications.

## References


[1] S.D. Brookes, C.A.R. Hoare, and A.W. Roscoe. A theory of communicating sequential processes. *Journal of the ACM*, 31(3):560–599, 1984.

[2] L. Cardelli. On process rate semantics. *Theor. Comput. Sci.*, 391(3):190–215, 2008.

[3] K. Chatzikokolakis and C. Palamidessi. A framework for analyzing probabilistic protocols and its application to the partial secrets exchange. *Theor. Comput. Sci.*, 389(3):512–527, 2007.

[4] G. Ciobanu. Software verification of biomolecular systems. In G. Ciobanu and G. Rozenberg, editors, *Modelling in Molecular Biology*, pages 40–59. Springer-Verlag, 2004.

[5] G. Ciobanu. From gene regulation to stochastic fusion. In *UC '08: Proceedings of the 7th international conference on Unconventional Computing*, pages 51–63, Berlin, Heidelberg, 2008. Springer-Verlag.

[6] U. Feige. A tight lower bound on the cover time for random walks on graphs. *RSA: Random Structures and Algorithms*, 6:51–54,433–438, 1995.

[7] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.

[8] J. Hillston. *A compositional approach to performance modelling*. PhD thesis, University of Edinburgh, 1994.

[9] J. Hillston and L. Kloul. An efficient Kronecker representation for PEPA models. In L. de Alfaro and S. Gilmore, editors, *Proceedings of the first joint PAPM-PROBMIV Workshop*, volume 2165 of *Lecture Notes in Computer Science*, pages 120–135, Aachen, Germany, September 2001. Springer-Verlag.

[10] R. Milner. *Communicating and mobile systems: the pi-calculus*. Cambridge University Press, 1999.

[11] C. Morgan and A. McIver. Cost analysis of games, using program logic. In *8th Asia-Pacific Software Engineering Conference (APSEC 2001)*, page 351 (Abstract only). Dec 2001.

[12] G. Norman, C. Palamidessi, D. Parker, and P. Wu. Model checking the probabilistic $\pi$-calculus. In *Proc. 4th International Conference on Quantitative Evaluation of Systems (QEST'07)*, pages 169–178. IEEE Computer Society, 2007.

[13] C. Priami, A. Regev, E. Y. Shapiro, and W. Silverman. Application of a stochastic name-passing calculus to representation and simulation of molecular processes. *Information Processing Letters*, 80(1):25–31, 2001.

# Poporo: A Formal Framework for Social Networking

**Nestor Catano**[1], **Vassilis Kostakos**[1], **Ian Oakley**[1]

[1] (ncatano,vk,ian)@uma.pt
Madeira-ITI, Portugal

**Abstract:** This position paper presents a novel approach for ensuring privacy in online social network applications through the combination of formal methods so as to reason in logic about privacy policies, graph theory and simulation to establish the potential threats of revealing information to users, and Human Computer Interaction to ensure that policies are usable and configurable.

**Keywords:** Social Networking, Security and Privacy Policies, Formal Methods, Refinement Calculus, Proof Carrying Code

## 1 Introduction

In recent years, online social network services in the form of websites such as Facebook, MySpace, LinkedIn and Hi5 have become popular tools to allow users to publish content, share common interests and keep up with their friends, family and business connections. A typical social network user profile features personal information (e.g. gender, birthday, family situation), a continuous stream of activity logged from actions taken on the site (such as messages sent, status updated, games played) and media content (e.g. personal photos and videos). The privacy and security of this information is a significant concern [GA05]. For example, users may upload media they wish to share with specific friends, but do not wish to be widely distributed to their network as a whole. Control of the access to the content on social network profiles is therefore an important issue. However, numerous tensions exist. For example, users find stipulating detailed privacy settings to be challenging and often fail to achieve their goals [BAC09]. Furthermore, social network services have conflicting goals. Although respecting the privacy of their client base is important, they must also grow and expand the connections between their users in order to be successful. This is typically achieved by exposing content to users through links such as "friends-of-friends", in which content relating to individuals known to a user's friends (but not the user) is revealed. Examples of this behaviour include gaining access to a photo album of an unknown user simply because a friend is tagged in one of the images.

This position paper argues that users need mechanisms to reliably restrict access to content in online social network services and suggests that formal methods [RV01] can provide a logical foundation with which to achieve this goal: to express and enforce privacy and security policies unambiguously. It outlines a vision in which social networking websites are used as a living test-bed for novel systems which combine formal methods, graph theory and Human-Computer Interaction (HCI) techniques to develop privacy and security systems which are secure, dependable, trustworthy and usable. These areas are discussed in the reminder of this paper.

# 2 Proposed Approach

The main components of Poporo are summarised as follows. We first construct Matelas[1], a predicate calculus abstract specification layer definition for social networking, modelling social-network content, privacy policies, social-networks friendship relations, and how these effect the policies with regards to content and other users in the network. Using refinement calculus techniques [HHS86], Matelas is ported into a social network core application that adheres to the stipulated policies. Using Proof Carrying Code (PCC) [Nec97], the functionality of this core is then extended by the development of plug-ins that adhere to the policies. These plug-ins are then automatically categorised in terms of their threat to privacy by analysing the APIs and graph algorithms utilised by each plug-in. Finally, HCI techniques are used to develop interfaces that effectively represent the policies of the core application to users, as well as enabling them to modify and adapt them to suit their preferences.

## 2.1 Matelas and Predicate Calculus

The basis of our work is Matelas, a specification layer that builds on predicate calculus and focuses on human centred privacy and security policies. Using refinement calculus techniques [HHS86], Matelas is used to construct a sound social network core application that adheres to stipulated policies. That is, from a predicate calculus specification of social networks, a code level specification model is attained while applying successive refinement steps.

We will deliver a social network core application that verifies and implements social network privacy policies considered in Matelas. The core application will serve as a common layer to which social network functionalities will be plugged-in. Matelas will distinguish four rather independent aspects of social networks, namely, user content and privacy issues, user content and how it is affected by friendship relations, the user interface, and the user content and its hierarchy.

## 2.2 Using Proof Carrying Code to Extend the Core Application

While the social network core application described in Section 2.1 is minimal in functionality, it will be considerably extended by incorporating plug-ins . This can be achieved by developing a framework where the plug-ins, written in popular programming languages such as Java or C, can demonstrate their adherence to the policies stipulated by Matelas. This will be achieved by using PCC [Nec97], which is a technique in which a code consumer (the social network core application) establishes a set of rules (privacy and security policies) that guarantee that externally produced programs (the plug-ins) can safely be run by the consumer. In addition to the code to be executed, the code producer must provide a proof of adherence to set of rules defined by the code consumer. This proof is run by the code consumer once, using a proof validator, to check whether the proof is valid and therefore the external program is safe to execute. It is imperative that the proof validator is automatic and fast. Hence, Matelas, while expressive enough for modelling general privacy and policy properties, must allow the (semi-) automatic checking of proofs.

[1] Matelas is the French word for the English word mattress.

The policies for Java plug-ins can be written in JML [LBR06] (Java Modeling Language), which allows different formal methods tools to check program correctness [BCHJ05, BCC+05]. JML specs have the advantage over predicate calculus based models in that they are close to Java, and thus are closer to average programmers. We therefore envisage to investigate on systematic ways JML specs can be translated into predicate calculus based models.

In summary, the main output of this work is a PCC based plug-in validator that checks plug-ins for compliance with Matelas defined policies, and a translation definition from Matelas to JML.

## 2.3 Formal Analysis of Social Networking Privacy

An important step in understanding the privacy threats of plug-ins is a rigorous assessment of the privacy threats posed by the various graph theory algorithms commonly used in social networking systems and services. For example, consider a plug-in that suggests new friends to a user, based on the user's existing friends and the relationships between those friends. One way to achieve this is to use a local clustering algorithm (e.g. transitivity or clustering coefficient). If such a plug-in has access to all of a user's friends, then it can statistically calculate the user's age by looking at his/her friends' age, location (similarly by looking at the locations of the user's friends), gender, work, etc. A first step in mapping the privacy threats of various graph algorithms provided by the social network core application is to run extensive simulations. Crucially, the underlying network structure may be important in determining the severity of the privacy threat. Therefore, a number of artificial social networks will be generated using a variety of parameters (such as size, density, degree distribution, average path length). These networks will be then populated with private and public information, such that they resemble a real-world social network. Our simulation environment will then use Monte-Carlo simulation of a number of graph algorithms (such as shortest path, calculation of betweenness, statistic features), to explore their privacy implications. Our formal specification engine will be used during these simulations to ensure that the algorithms adhere to privacy policies.

We will deliver a classification of graph theory algorithms based on the threat to privacy. This classification can be used to automatically assign a simple "Privacy" label (e.g. Green, Yellow, Red) to 3rd party software or plug-ins that will be used by our formal specification engine. This label is intended to communicate to users the threat to their privacy, in a simple and understandable fashion, much in the spirit of Section 2.4.

## 2.4 Human Considerations

While a social network application or plug-in may provably adhere to policies, these policies are typically sufficiently abstract to allow for human error. While a plug-in may not access users date of birth without explicit authorisation, it is still possible for users to inadvertently give such authorisation. This may happen either by accident or, most likely, due to the complexity of the settings and preferences interface that the user is asked to interact with. Hence it is imperative to augment the provably correct social network core and plug-ins with understandable human interfaces that enable end users to express their privacy policies and preferences, as well as to review and modify them. This can be achieved by a number of approaches. First, providing clear and understandable labels and metaphors that effectively present policies. Second, enabling

users to interact with their policies, obtaining feed-forward about the potential effects of any changes they make to their policies. To this end, we will use iterative design and testing of initial sketches and prototypes. In addition, we will perform heuristic analyses and cognitive walk-throughs to identify potential problems with our human interfaces. Finally, we will carry out user observations to measure users' reaction to our designs as well as their subjective preferences.

We will deliver a set of prototype designs that will be integrated into our social networking system (possibly developed as a plug-in). These designs will be responsible for acting as a bridge between end users and our formal specification engine.

## 3 Conclusion

This position paper has presented a novel vision of maintaining privacy and securing data in online social network services through the combination of formal methods (to provide provable behaviours), simulation and graph theory (to provide meaningful generalisations of algorithmic specified activities) and HCI (to ensure that systems are usable and allow individuals to quickly and effectively configure them). We believe that this approach will lead to the development of new back-end systems, front-end prototypes and theoretical understanding relevant to the complex issues underlying the security and privacy of data stored in online social network services.

## Bibliography

[BAC09] J. Bonneau, J. Anderson, L. Church. Privacy Suites: Shared Privacy for Social Networks. In *Symposium on Usable Privacy and Security (SOUPS)*. July 2009.

[BCC+05] L. Burdy, Y. Cheon, D. Cok, M. Ernst, J. Kiniry, G. T. Leavens, K. R. M. Leino, E. Poll. An Overview of JML Tools and Applications. *International Journal on Software Tools for Technology Transfer (STTT)* 7(3):212–232, June 2005.

[BCHJ05] C. Breunesse, N. Catano, M. Huisman, B. Jacobs. Formal Methods for Smart Cards: An Experience Report. *Science of Computer Programming* 55(1-3):53–80, March 2005.

[GA05] R. Gross, A. Acquisti. Information Revelation and Privacy in Online Social Networks. In *Workshop on Privacy in the Electronic Society (WPES)*. Pp. 71–80. 2005.

[HHS86] J. He, C. A. R. Hoare, J. W. Sanders. Data Refinement Refined. In *European Symposium on Programming (ESOP)*. Pp. 187–196. 1986.

[LBR06] G. Leavens, A. Baker, C. Ruby. Preliminary Design of JML: A Behavioral Interface Specification Language for Java. *ACM SIGSOFT Software Engineering Notes* 31(3):1–38, 2006.

[Nec97] G. C. Necula. Proof-Carrying Code. In *Symposium on Principles of Programming Languages (POPL)*. P. 106119. Paris, January 1997.

[RV01] A. Robinson, A. Voronkov. *Handbook of Automated Reasoning*. MIT Press, 2001.

# Operational Model: Integrating User Tasks and Environment Information with System Model

**Sébastien Combéfis**

Computer Science and Engineering Department
Université catholique de Louvain
Place Sainte Barbe, 2
1348, Louvain-la-Neuve, Belgium
Sebastien.Combefis@uclouvain.be

**Abstract:** This paper addresses the problem of integrating information about user tasks and about the operating environment to the model of a system. Following a modelling based on labelled transition systems, this integration can be done with elementary operations: models synchronization and graph operations. Integration of user tasks and information about operating environment allows to get operational model which represents the knowledge the user should have about the system to perform a set of tasks, given information provided by the operating environment, through user interface for example. The paper draws up a formal way to do the integration to get an operational model which can be used to evaluate and compare different system's design, to do verification or to generate training material.

**Keywords:** Formal methods, Human-Computer Interaction, Task-System Integration, Parallel synchronization, User tasks, System Design Evaluation and Comparison

## 1 Introduction

When analyzing and verifying the design and the specification of a system, besides the system itself, the integration of user tasks plays an important role. The user tasks which represents what the user wants to do with the system — his goals — can change completely the way a system is conceived and designed. There are a lot of whole community using formal methods to analyse Human-Computer Interaction (HCI), providing rigorous, systematic and automated ways of analysis. Information about user tasks is used to help the analysis of specification [Cam03, PS01] or to analyse the effects of erroneous human behaviour by building deviations in how tasks are performed by the user [PS02, BB07].

Integration between tasks and system has been studied these years, see for example [NPP+01]. This paper proposes to integrate information about user tasks and operating environment with the description of the system. This integration provides an *operational model* of the system which represents part of the full behaviour of the system that is relevant according to the user tasks and the operation environment. The main contribution is the definition of a new model, combining information about the system, the tasks and the operation environment. Once computed, this model can be used to perform various analysis.

## 2 Operational Model

The *operational model* of a system only covers the behaviour part of the system model which is relevant for the user. Indeed, the user do not need to know the full behaviour of the system to use it because they only want to perform some *tasks* with it. Moreover, when operating the system, the user gets some feedback from it, that is precisely what we call the *operating environment*. That feedback may help the user to operate the system. Figure 1 illustrates the components the operational model depends on: the *system model* represents the full behaviour of the system, the *tasks model* represents the tasks that the user should be able to perform on the system and the *action-based user interface* represents part of the user interface, that is information about which event is occuring in the system.

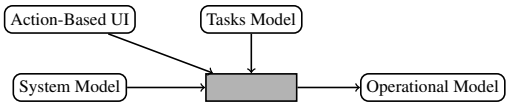


Figure 1: Computing operational model from system and tasks models.

The modelling approach follows the one of [CP09]. The system is modelled as a *labelled transition system* (LTS) $\mathscr{M} = \langle S, \mathscr{L}, s_0, \rightarrow \rangle$ with $S$ the set of states, $\mathscr{L}$ the set of actions, $s_0$ the initial state and $\rightarrow \subseteq S \times \mathscr{L} \times S$ the transition relation. The $\tau$ action denotes an *internal action* representing all transitions that are not observable from outside the system. The *action-based user interface* consists in a distinction among observable actions: *commands* are performed by the user on the system and *observations* are controlled by the system and occur autonomously without any user action. The system model is considered as given and is built by the system's designers.

### 2.1 Integrating Tasks

User tasks are modelled using ConcurTaskTrees [PMM97] which is a graphical notation that allows designers to describe hierarchies of tasks linked with temporal relations following the semantic of LOTOS [ISO89]. To fit in our framework, all the user and application tasks that are leaves should come from $\mathscr{L}$, user tasks corresponding to commands and application tasks to observations. To be able to integrate user tasks with the system model, they are transformed from CTT descriptions to a set of LTSs $\{\mathscr{T}_1, \cdots, \mathscr{T}_n\}$ which can be done in an automated way [PS01].

The integration of user tasks with the system is achieved with a *synchronous parallel composition* between the system model $\mathscr{M}$ and the tasks model $\mathscr{T}$, synchronized on $\mathscr{L} \setminus \{\tau\}$. The tasks model $\mathscr{T}$ is built from the set $\{\mathscr{T}_1, \cdots, \mathscr{T}_n\}$ of user tasks, adding transitions $s_{0_\mathscr{T}} \xrightarrow{\tau} \mathscr{T}_i$ for each user task $\mathscr{T}_i$, where $s_{0_\mathscr{T}}$ is the initial state of $\mathscr{T}$. The model $\mathscr{M} \parallel \mathscr{T}$ can contains internal transitions which are not relevant for the user; they are removed using the *edge contraction* operation from graph theory.

### 2.2 Integrating Environment Information

Information about the operating environment can be integrated by performing two simplifications on the system model. The first simplification that can be done comes from the idea that the user is not obliged to know all the paths of the system exactly. Some paths can indeed be ignored by the user provided that the user knows that the path has been followed, for example through the user interface. The second simplification comes from the hierarchy in the user tasks; the user should not be required to know the system at the finer level, some actions can be gathered into one single action.

Figure 2 illustrates the two simplification. The system modelled is a vending machine accepting credit card or cash. If the user selects creditCard, after encoding his/her PIN number, the system will enter a procedure to contact the bank. The progress is shown by the user interface through observations that allow the user to follow the transaction. All the states from the one preceding the connected action to the one after the dataRecv action can be merged into a single state, preserving any edges linked to any of the merged states. The second kind of simplification can be done according to the tasks hierarchy. Here, the actions 5in and 10in refine addMoney. All these actions can be renamed and states linked with the addMoney action can be merged because the user does not need to distinguish the 5in and 10in anymore.

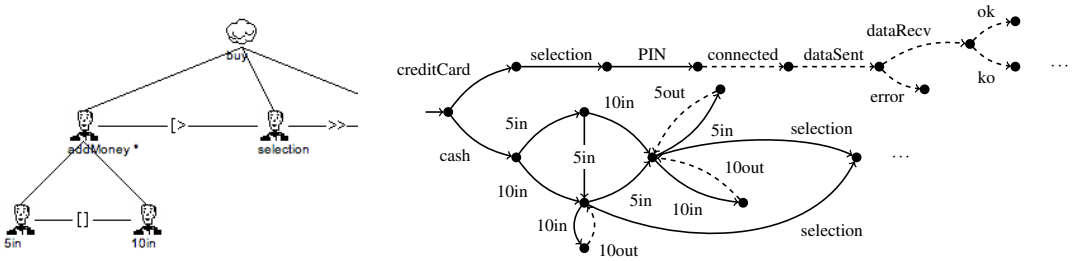


Figure 2: Example of a user task (left) and of a system model illustrating the integration of environment information (right). The system is a vending machine accepting payment with credit card or cash. Plain lines represent commands and dashed lines represent observations.

## 3 Conclusion and Perspectives

This paper defines the notion of *operational model* of a system. An operational model captures the part of the behaviour of a system which is relevant to a user who should be able to perform some tasks and gets information about the operation environment through the user interface. Such a model can be used for different purposes: generation of training user manual, evaluation and comparison of different systems, checking properties, ... Another possibility is to check whether existing manual's content concurs with the knowledge needed for the user. Viewing a manual as a set of scenarios, it is possible to generate a corresponding model [DLDL05] and then to compare it with the operational model.

Further work includes implementing a prototype for the computation of an operational model for a given system. Another direction consists in using operational models to evaluate and com-

pare system models based on a set of user tasks, assessing the usability of the system using a metric on the operational model.

**Acknowledgements:** This work is partly supported by project MoVES under the Interuniversity Attraction Poles Programme — Belgian State — Belgian Science Policy.

## Bibliography

[BB07] R. Bastide, S. Basnyat. Error Patterns: Systematic Investigation of Deviations in Task Models. In Coninx et al. (eds.), *Proceedings of the 5th International Workshop on Task Models Diagrams for UI Design*. Lecture Notes in Computer Science 4285, pp. 109–121. Springer-Verlag, 2007.

[Cam03] J. C. Campos. Using task knowledge to guide interactor specifications analysis. In *Proceedings of the 10th International Workshop on Design, Specification and Verification of Interactive Systems*. Lecture Notes in Computer Science 2844, pp. 171–186. Springer-Verlag, 2003.

[CP09] S. Combéfis, C. Pecheur. A Bisimulation-Based Approach to the Analysis of Human-Computer Interaction. In Calvary et al. (eds.), *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. Pp. 101–110. ACM, New York, NY, USA, 2009.

[DLDL05] C. Damas, B. Lambeau, P. Dupont, A. van Lamsweerde. Generating Annotated Behavior Models from End-User Scenarios. *IEEE Transactions on Software Engineering* 31(12):1056–1073, Dec. 2005.

[ISO89] ISO 8807:1989. *Information processing systems – Open Systems Interconnection – LOTOS – A formal description technique based on the temporal ordering of observational behaviour*. International Organization for Standardization, Geneva, Switzerland, 1989.

[NPP+01] D. Navarre, P. Palanque, F. Paternò, C. Santoro, R. Bastide. A Tool Suite for Integrating Task and System Models through Scenarios. In *Proceedings of the 8th International Workshop on Design, Specification and Verification of Interactive Systems*. Lecture Notes in Computer Science 2220, pp. 88–113. Springer-Verlag, 2001.

[PMM97] F. Paternò, C. Mancini, S. Meniconi. ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. In *Proceedings of the IFIP TC13 International Conference on Human-Computer Interaction*. Pp. 362–369. Chapman & Hall, Ltd., London, UK, 1997.

[PS01] F. Paternò, C. Santoro. Integrating Model Checking and HCI Tools to Help Designers Verify User Interface Properties. In *Proceedings of the 7th International Workshop on Design, Specification and Verification of Interactive Systems*. Lecture Notes in Computer Science 1946, pp. 135–150. Springer-Verlag, 2001.

[PS02] F. Paternò, C. Santoro. Preventing User Errors by Systematic Analysis of Deviations from the System Task Model. *International Journal of Human-Computer Studies* 56(2):225–245, Feb. 2002.

# Roadmap for a Formal Approach to Reduce Inconsistencies in Enterprise Architecture Views

**Sietse Overbeek**[1], **Antonio Cerone**[2], **Marijn Janssen**[3]

[1]S.J.Overbeek@tudelft.nl,[3]M.F.W.H.A.Janssen@tudelft.nl
Faculty of Technology, Policy and Management, Delft University of Technology, Jaffalaan 5, 2600 GA Delft, The Netherlands

[2]antonio@iist.unu.edu
International Institute for Software Technology, United Nations University, Casa Silva Mendes, Est. do Engenheiro Trigo No. 4, Macau SAR China

**Abstract:** Enterprise architecture (EA) refers to a comprehensive description of all the key elements and structural as well as behavioral relationships that make up an enterprise. Enterprise architecting is aimed at matching the business processes and goals of an enterprise, together with software applications and information systems as well as human processes that are present in the enterprise. Contemporary EA frameworks incorporate viewpoints that can be utilized to exactly represent that part of an enterprise from the perspective or 'view' that is of special interest for a stakeholder. Such views are informally described using visual notations, usually with no associated semantics, or natural language to represent a part of the enterprise. In fact, a view is interpreted from the perspective of an individual stakeholder. This may create inconsistency in the interpretation of a view. In this paper the ArchiMate framework that is currently widely employed is analyzed to determine how views for distinct stakeholders are presented. This results in a roadmap for creating a future formal approach to define view presentations.

**Keywords:** Cognitive characteristics, Enterprise architecture, Formal methods, Stakeholders, Views

## 1 Introduction

The notion of enterprise architecture (EA) refers to a comprehensive description of all the key elements and relationships that make up an enterprise [Gui09]. In this definition, an enterprise may be a company, an institution, or a department within a company or an institution. The elements to be described may be data, network equipments, software components, business locations, human resources, and so on. Enterprise architecting is aimed at matching the business processes and goals of an enterprise, together with software applications and information systems as well as human processes that are present in the enterprise. An enterprise architecture is necessary, but not sufficient for successful implementation. The use of enterprise architecture requires that there exists strong communication, coordination, and cooperation between ICT and business personnel. Only in this way can the architecture become a *shared* architecture. The

importance of communication in achieving project success has been well documented in technology adoption literature [CJGM01]. Lack of communication has been linked to numerous project failures [PP99]. Information about the architecture flows to organizational members by communicating, requiring unambiguity and a shared meaning.

Contemporary EA frameworks incorporate viewpoints that can be utilized to exactly represent that part of an enterprise from the perspective or 'view' that is of special interest for a stakeholder. A view can be defined to be a representation of a system from the perspective of a related set of concerns [IEE00]. A stakeholder can be an individual, team, or organization (or classes thereof) with interests in, or concerns relative to, a system [IEE00]. A viewpoint is defined as a specification of the conventions for constructing and using a view; it is therefore a pattern or template from which to develop individual views by establishing the purposes and audience for a view and the techniques for its creation and analysis. Views are often informally described using visual notations, usually with no associated semantics, or natural language to represent a part of the enterprise. In fact, a view is interpreted from the perspective of an individual stakeholder. This may create inconsistency in the interpretation of a view. In this paper the ArchiMate framework is analyzed to determine how views for distinct stakeholders are presented. Based on these findings, a roadmap for a future formal language to define view presentations are introduced.

## 2 Analysis of view presentations in ArchiMate

Viewpoints and views are defined and classified in ArchiMate [The09] to help an architect in selecting the right conventions for the task at hand. This classification is based on two dimensions: purpose and content. The following three types of architecture support the purpose dimension: 1) Design viewpoints support architects in the design process from a draft to final design. These viewpoints consist of diagrams such as UML diagrams. 2) Decision support viewpoints assist managers in decision-making by providing insight into architecture relations. Examples are cross-reference tables, landscape maps, and lists. 3) Informing viewpoints assist to notify any stakeholder about the EA to achieve understanding and commitment, and to convince objectors. Examples are illustrations, animations, flyers, and so on. For characterizing the content of a view the following levels are defined: 1) Views on the detail level typically consider one layer and one aspect from the ArchiMate framework. A typical stakeholder is a software engineer responsible for design and implementation of a software component. An example view is an ER diagram. 2) At the coherence abstraction level, multiple layers or aspects are spanned. Extending the view to more than one layer or aspect enables the stakeholder to focus on relations like process-uses-system (multiple layer) or application-uses-object (multiple aspect). Typical stakeholders are operational managers responsible for a collection of IT services. 3) The overview level addresses both multiple layers and aspects. Such overviews are addressed to enterprise architects and decision makers.

## 3 Roadmap for a formal language to define view presentations

With the help of the viewpoint purpose and content classification provided by ArchiMate, it is easier to find typical conventions that might be useful in a given situation. However, an orthog-

onal categorization of each viewpoint into one of the purpose categories and content categories is not provided by ArchiMate. The six mentioned viewpoint categories are not exclusive. In fact, a viewpoint in one category cannot be applied to achieve another type of support. For instance, some decision support viewpoints may be used to communicate to any other stakeholder as well. The ArchiMate language uses visual notations for elements that can be found in an enterprise. These visual notations are also used to model the viewpoints and the views that are the result of applying one or more viewpoints. A summary of these visual notations can be found in [The09]. ArchiMate provides a natural language explanation for each of these symbols. For example, an actor is explained to be 'an organizational entity capable of (actively) performing behavior' [The09]. Because only a natural language description is offered, inconsistency in the interpretation of a view may occur when a view is interpreted by a stakeholder.

A formal approach should facilitate detection and elimination of inconsistencies within specific views of an architecture description and between views corresponding to distinct stakeholders. This disambiguates communication and enables coordination and cooperation among stakeholders. The semantics of a view presentation can be given as a collection of interpretations by different categories of stakeholders. Stakeholder categorization can be achieved in terms of their *cognitive characteristics*, resulting in a *cognitive setting*. This can be modeled as follows:

$$\mathsf{setting} : \mathsf{StkhTypes} \rightarrow \wp(\mathsf{CognChars}) \tag{1}$$

The expression $\mathsf{setting}(t) = C$ shows that a stakeholder of type $t \in \mathsf{StkhTypes}$ has the cognitive characteristics in set $C \subseteq \mathsf{CognChars}$. In [Ove09], a formal categorization of actors has been made based on cognitive characteristics. This categorization has been used to understand what type of actors supply which cognitive characteristics when performing knowledge-intensive tasks. Examples of such characteristics can be the ability to fulfill a task on your own (*independency*), the willpower to fulfill a task (*volition*), the ability to transform knowledge types (*causability*), and cognitive improvements (*improvability*). Once we understand which cognitive characteristics belong to which stakeholder type, we can also understand which collection of viewpoints a stakeholder of a certain type uses in understanding a view. This can be formalized as follows:

$$\mathsf{interpret} : \mathsf{Views} \rightarrow (\mathsf{StkhTypes} \rightarrow \wp(\mathsf{Viewpoints})) \tag{2}$$

The expression $\mathsf{interpret}_v(t)$ is the set of viewpoints $V \subseteq \mathsf{Viewpoints}$ that a stakeholder type $t \in \mathsf{StkhTypes}$ uses in understanding view $v \in \mathsf{Views}$. Here, Views is the set of views, StkhTypes is the set of stakeholder types, and Viewpoints is the set of viewpoints. If for a stakeholder of type $t$, view $v$ is not relevant then $\mathsf{interpret}_v(t) = \emptyset$. If stakeholders of types $t_1$ and $t_2$ cooperate in performing a task using views $v_1$ and $v_2$ then communication between $t_1$ and $t_2$ is not possible if $\mathsf{interpret}_{v_1}(t_1) \cap \mathsf{interpret}_{v_2}(t_2) = \emptyset$ and this may create an inconsistency. However, cognitive characteristics of one of the two stakeholders may lead to a dynamic resolution of the inconsistency, i.e. *independency* and *volition* characteristics may allow one of the stakeholders to take the lead or even complete the task on his own. *Causability* in one stakeholder may cause the expansion of the set of relevant viewpoints in the other stakeholder. *Improvability* facilitates the expansion of the set of relevant viewpoints in one of the stakeholders, and so on. Of course, possible viewpoints have to be identified and formally defined as well in order to understand their

meaning. These interpretations can be at least partially found by studying existing viewpoints in contemporary EA frameworks, such as the discussed viewpoints of the ArchiMate framework. Once possible viewpoints are identified and coupled to stakeholder types, it is possible to formalize the viewpoints for those parts where interpretation ambiguity may occur. For example, non-formalized views that are based on non-formalized viewpoints may consist of UML diagrams, may contain use case descriptions in natural language, or may include ArchiMate visuals. By using the formalized viewpoints that are relevant for a stakeholder type, it is possible to identify which parts of a view are expressions of formalizations found in the viewpoint. In other words, viewpoints can be translated to mutually exclusive formal frameworks and the views are expressions of them for a given situation.

## 4 Conclusions

Contemporary EA frameworks incorporate viewpoints that can be utilized to exactly represent a part of an enterprise from the view that is of interest for a stakeholder. Such views are informally described and interpreted from the perspective of an individual stakeholder. Because this may create inconsistency in the interpretation of a view, we have presented a roadmap for creating a future formal language to define view presentations: 1) Identification of stakeholder types; 2) Categorization of stakeholder types by means of cognitive characteristics; 3) Group viewpoints to stakeholder types; 4) Formalize those parts of the viewpoints that remain ambiguous, and finally: 5) Use the viewpoint formalizations to generate expressions that are the resulting views.

## Bibliography

[CJGM01] F. Carter, T. Jambulingham, V. Gupta, N. Melone. Technological innovations: a framework for communicating diffusion effects. *Information & Management* 38(5):277–287, 2001.

[Gui09] L. Guijarro. Semantic interoperability in eGovernment initiatives. *Computer Standards & Interfaces* 31(1):174–180, 2009.

[IEE00] IEEE. IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. Technical report IEEE Std 1471-2000, IEEE Computer Society, New York, NY, USA, 2000.

[Ove09] S. Overbeek. *Bridging Supply and Demand for Knowledge Intensive Tasks: Based on Knowledge, Cognition, and Quality*. PhD thesis, Radboud University Nijmegen, The Netherlands, 2009.

[PP99] M. Pinto, J. Pinto. Project team communication and cross-functional cooperation in new program development. *Journal of Product Innovation Management* 7(3):200–212, 1999.

[The09] The_Open_Group. ArchiMate 1.0 Specification. Technical report C091, The Open Group, Reading, United Kingdom, 2009.