Proceedings of
1st Open EIT ICT Labs Workshop
on
Cyber-Physical Systems Engineering

Fondazione Bruno Kessler – Trento

May 24, 2013

Editors

María Victoria Cengarle (fortiss GmbH, Germany)
Marco Roveri (FBK, Italy)

Preface

This volume contains the contributions to the First Open EIT ICT Labs Workshop on Cyber-Physical Systems Engineering, held on May 24, 2013, in Trento, Italy (EIT CPSE 2013).

The ongoing integration of software-intensive embedded systems and global communication networks into Cyber-Physical Systems (CPS) is considered to be the next revolution in ICT with a great deal of game-changing business potential and novel business models for integrated products and services. Many technology leaders are already in the midst of a global race of repositioning and reinventing themselves by developing new dynamic CPS-inspired business models.

Mastering the engineering of complex and trustworthy CPS is crucial to implementing such business models. Current systems engineering frameworks, however, do not enable a conceptualization and design for the deep interdependencies among engineered systems and the natural world. Thus, there is a clear need for a new Cyber-Physical Systems Engineering (CPSE) framework for apprehending the development and operation of highly efficient CPS upon which people can almost blindly depend.

The aim of the workshop was to provide researchers and practicioners working in the realm of CPS with a forum for discussion and interchange. The topics suggested for the workshop included

- integrated processes based on the design-operation life-cycle continuum,
- multidisciplinary modelling and analysis of discrete-continuous systems,
- composition and integration frameworks for heterogeneous systems,
- scalable and evolvable system and software architectures,
- open and standardized tool integration frameworks for performance and trustworthiness analysis,
- industrial experience on engineering CPS;

the selection procedure albeit was open to consider any further question related to this subject of disruptive potential.

The Program Committee selected 7 submissions (from 16 different countries) of high scientific quality, originality, and relevance to the workshop. Each paper was reviewed by at least three Program Committee members or external referees. Additionally, the workshop invited two outstanding researchers to present their advances in the area, namely Bill Roscoe (University of Oxford, UK) and Radu Calinescu (University of York, UK).

María Victoria Cengarle (fortiss GmbH, Germany)
Marco Roveri (FBK, Italy)

Program Committee

Manfred Broy
María Victoria Cengarle
Alessandro Cimatti
Rainer Ersch
Elena Fersman
Cornel Klein
Luigi Palopoli
Holger Pfeifer
Marco Roveri
Harald Ruess
Martin Törngren

Additional Reviewers

Bauer, Veronika
Chen, Dejiu
El-Khoury, Jad
Hölzl, Florian
Mou, Dongyue
Schorp, Konstantin

Table of Contents

Automatic timewise renement ....
Bill Roscoe

Runtime Quantitative Verification: Applications and Research Challenges ....
Radu Calinescu, Simos Gerasimou, Kenneth Johnson and Yasmin Rafiq

Towards autonomous embedded systems ....
Sagar Behere, Martin Törngren, Jad El-Khoury and Dejiu Chen

Synchronous Specialization of Alf for Cyber-Physical Systems ....
Alessandro Romero, Klaus Schneider and Mauricio Gonçalves Vieira Ferreira

Robotic car-like vehicles: a case study for cyberphysical systems ....
Luigi Palopoli, Federico Moro, Daniele Fontanelli and Tizar Rizano

Towards Dynamic Deployment Calculation for Extensible Systems using SMT-Solvers ....
Klaus Becker and Sebastian Voss

On the Technological and Methodological Concepts of Federated Embedded Systems .....
Avenir Kobetski and Jakob Axelsson

A Roadmap Towards Integrated CPS Development Environments ....
Jad El-Khoury, Fredrik Asplund, Matthias Biehl, Frederic Loiret and Martin Törngren

Engineering of Cyber-Physical Systems ....
María Victoria Cengarle

The automated verification of timewise refinement (Draft)

A.W. Roscoe,

Oxford University Department of Computer Science

Abstract. While Hoare's CSP models reactive systems without assigning an exact time to events, Timed CSP records the exact times as nonnegative reals. Timed CSP therefore provides a more exact semantics of systems, but it still makes sense to ask whether a timed process satisfies an untimed specification. Indeed the question of whether such specifications are satisfied often reduces to the timing details of the implementation. Schneider showed how this could be understood at an abstract level via the concept of timewise refinement. The recent implementation of Timed CSP in the CSP refinement checker FDR (using Ouaknine's theory of digitisation) has at last provided the framework for automating timewise refinement. In this paper we show how to do this, discovering that it is subtle because of the need to reconcile infinite behaviours with finite ones

1 Introduction

Hoare's process algebra CSP [5, 11, 12] provides a framework for describing systems patterns of actions representing communications. It provides operators that allows the description of implementations, typically consisting of parallel networks whose parts communicate by handshaking on their own action. It also provides the means – including representations of nondeterminism, deadlock, and other pathologies – for creating specifications.

The semantics of CSP in its original form cared about the order in which actions happen, but not their exact times. The author and Mike Reed [8] introduced an alternative view of the same notation, with the addition of the timed constant process $WAIT\ t$ that terminates successfully after t time units.

Substantial theories have been developed for both the untimed and timed versions of CSP including abstract models and operational semantics. In the timed models one records more details than in the untimed ones, and so one would expect to have mappings which forget this extra detail and therefore cast any system modelled in timed theories to values in corresponding untimed ones.

Timed systems may be expected to meet specifications which restrict the times at which events happen, as well as ones that do not. In the latter case the correctness of the system may well depend on timing details within a system even though the specification is untimed.[1] For such cases we would need the

[1] A good example of this is provided by the level crossing described in [3, 11], where the untimed specification that the gate is down when a train passes by it is only

detail of Timed CSP to create a model that establishes the specification, and then to have a way of testing it against an untimed specification.

For untimed trace, or safety, specifications, there can be no doubt what this means since there can be no doubt how to extract an untimed trace from a timed behaviour: simply extract the sequence of events that occur in it.

Beyond traces, most used untimed models for CSP are *failures* and *failures-divergences*. Each of these depends, in the usual operational formulation, on the dual ideas of divergence (an unbroken infinite series of internal τ actions and *stability* (reaching a state where no τ action is possible). While it is possible to build timed models that capture this information such as the timed failures-stability model [9], it is far from clear that this provides the most useful link between Timed CSP the untimed failures model.

One argument for this is that divergence is a much less dangerous and difficult phenomenon in the timed world than it is in the untimed one. On the "no-Zeno" assumption that only finitely many events happen in a finite time, divergence can only occur over an infinite interval, during which we can see the offers that the process makes in a way that makes no sense over untimed models. Whereas refusal sets only appear in the untimed models when they become permanent through stability, in the timed models there is no choice but to record what is refused at every instant – event when a τ will later become enabled from the current state.

Schneider [15, 16] therefore described the *timewise refinement* relation between the failures model of untimed CSP and the timed failures model, in which events are given times from the non-negative real numbers, and we record what was (observed to have been) refused at each moment of time. $P_{SF} \sqsubseteq_{TF} Q$[2] if all of the untimed traces that can be extracted from Q are permissible for P, and furthermore whenever Q can perform the untimed trace s and from some time beyond the end of s always refuse the whole of the set X of visible actions, then (s, X) is a failure of P.

Thanks to Ouaknine's work on *digitisation* [6, 7], it has become possible to establish results about (continuous) Timed CSP processes[3] by proving them about a version of Timed CSP formulated over discrete time, which in turn is equivalent to proving analogous results about a *tock*-CSP translation of the discrete version. *tock*-CSP [11] is ordinary "untimed" CSP but with the special event *tock* interpreted as the regular passage of time. This translation was described in [6], and its implementation in the FDR refinement checker in [1].

That made it straightforward to check untimed *trace* properties of Timed CSP processes on FDR: all one had to do was to check untimed trace refinement of the untimed specification P by $Q \setminus \{tock\}$ for the timed implementation Q. In other words we can simply hide the special time event.

---

met because, inter alia, of the relationship of the speed of trains to the timing characteristics of the physical gate.

2 This notation is borrowed from [16].

3 These results are restricted to processes in which all programmer- and implementation-created delays are of integer length: *integer* Timed CSP.

The situation is not so simple for failures, since the permanent refusal of a set of events in a *tock*-time process necessarily involves an infinite series of *tock*s and quite possibly states that vary throughout time. In particular $Q \setminus \{tock\}$ will have no stable failures at all, since no discrete Timed CSP process ever refuses *tock*. Therefore the inclusion of Timed CSP in FDR did not in itself solve the problem of automating the question of timewise refinement.

Timed CSP's semantics have the property known as *maximal progress*, meaning that a process cannot do nothing when a τ action is possible. In the translation to *tock*-CSP this translates to the statement that no *tock* action can happen when a τ is possible. Another way of saying the same thing is that τ must have priority over *tock*.

For this and other reasons a priority operator has been implemented in FDR, as proposed in [12]. We shall see in the present paper that priority also provides the solution to the problem of characterising timewise refinement, using a similar idea to that used to solve a problem of abstraction in [13]. Specifically, we will find that way to map a Timed CSP process to a value that can be compared against an untimed failures specification is to use the *slow abstraction* defined in that paper.

This, in the present case, has the effect of turning any refusal (as opposed to trace) counter-example to a failures refinement into a divergence that can be detected by FDR.

The rest of this paper is organised as follows. In the next (background) section we recall the essential details of CSP, Timed CSP and their models, giving the formal definition of timewise refinement. In Section 4 we shown how slow abstraction of $\{tock\}$ provides an accurate characterisation of timewise refinement for the discrete interpretation of Timed CSP, with digitisation extending this result to the continuous interpretation.

In this paper we make the assumption that the underlying alphabet of actions Σ is finite. This is necessary because of the way we check timewise refinement later, and is common to [13]. Since FDR can only handle finite alphabets, this is therefore no restriction at all at that level.

2 Background

2.1 CSP and its semantics

In this section we give an overview of untimed CSP and its models, as relevant to the present paper. Far more extensive presentations can be found in [5, 11, 12]. CSP is based on instantaneous actions handshaken between a process and its environment, whether that environment consists of processes it is interacting with or some notional external observer. It enables the modelling and analysis of patterns of interaction. The books [5, 11, 12, 16] all provide thorough introductions to CSP. The main constructs that we will be using in this paper are set out below.

- The processes $STOP$, $SKIP$ and **div** respectively do nothing, terminate immediately with the signal ✓ and diverge by repeating the internal action τ. Run_A and $Chaos_A$ can each perform any sequence of events from A, but while Run_A always offers the environment every member of A, $Chaos_A$ can nondeterministically choose to offer just those members of A it selects, including none at all.
- $a \rightarrow P$ *prefixes* P with the single communication a which belongs to the set Σ of normal visible communications. Similarly $?x : A \rightarrow P(x)$ offers the choice A and then behaves accordingly.
- CSP has several *choice* operators. $P \mathrel{\Box} Q$ and $P \sqcap Q$ respectively offer the environment the first visible events of P and Q, make an internal decision via τ actions whether to behave like P or Q.
  The asymmetric choice operator $P \rhd Q$ offers the initial visible choices of P until it performs a τ action and opts to behave like Q. In the cases of $P \mathrel{\Box} Q$ and $P \rhd Q$, the subsequent behaviour depends on what initial action occurs.
- $P \setminus X$ (hiding) behaves like P except that all actions in X become (internal and invisible) τs.
- $P[\![R]\!]$ (renaming) behaves like P except that whenever P performs an action a, the *renamed* process must perform some b that is related to a under the relation R.
- $P \underset{A}{\parallel} Q$ is a *parallel* operator under which P and Q act independently except that they have to agree (i.e. synchronise or handshake) on all communications in A. A number of other parallel operators can be defined in terms of this, including $P \mathbin{|||} Q = P \underset{\emptyset}{\parallel} Q$ in which no synchronisation happens at all.
- $P; Q$ behaves like P until it terminates successfully, and then like Q.

There are also other operators such as $P \bigtriangleup Q$ (interrupt) and $P\, \Theta_a\, Q$ (throwing an exception) that do not play a direct role in this paper.

It is always asserted that the meaning, or semantics, of a CSP process is the pattern of externally visible communication it exhibits. As shown in [11, 12], CSP has several styles of semantics, that can be shown to be appropriately consistent with one another. The two styles that will concern us are *operational* semantics, in which rules are given that interpret any closed process term as a labelled transition system (LTS), and *behavioural* models, in which processes are identified with sets of observations that might be made from the outside.

An LTS models a process as a set of states that it moves between via actions in Σ^τ, where τ cannot be seen or controlled by the environment. There may be many actions with the same label a single state, in which case the environment has has no control over which is followed. The best known behavioural models of CSP are based on the following types of observation. *Traces* are sequences of visible communications a process can perform. *Failures* are combinations (s, X) of a finite trace s and a set of actions that the process can refuse in a *stable* state reachable on s. A state is stable if it cannot perform τ. *Divergences* are traces after which the process can perform an infinite uninterrupted sequence of τ actions, in other words diverge. The models are then

- $\mathcal{T}$ in which a process is identified with its set of finite traces;
- $\mathcal{F}$ in which it is modelled by its (stable) failures and finite traces;
- $\mathcal{N}$ in which it is modelled by its sets of failures and divergences, both extended by all extensions of divergences: it is *divergence strict*.

2.2 Timed CSP and its semantics

For thorough presentations of Timed CSP and its models, the reader should study [16, 8, 9].

Timed CSP has the same operators as CSP, plus a single constant with explicit time: $\mathit{WAIT}\ t$ where t is a non-negative number. This terminates with ✓ t time after it is started. More operators such as timeout and timed interrupt can be defined in terms of the basic ones. The difference between the interpretation of operators in CSP and Timed CSP is that in the latter we are precise about when communications happen and become available, while remaining faithful to the understanding contained in the untimed interpretation. So, for example the process $a \to P$ still offers the event a until it occurs. P then starts some fixed time (representing the time it takes the process to recover from, or perhaps complete, a) before P starts. And $P \,\square\, Q$ still offers the choice of the initial (visible) events of P and Q until one of them performs one. Thinking about this makes it clear that P and Q evolve in time – perhaps performing their own τ events – side by side until one of them has an action accepted, allowing the other to be turned off.

Untimed CSP carries the assumption that an unstable state – one in which a τ action is enabled – cannot persist, with a τ or some other action definitely happening quickly. In Timed CSP this has to be quantified, leading to the concept of *maximal progress*: when a τ is enabled, some action must happen *immediately*. This simple and seemingly straightforward translation of intuition has a major effect on the semantic models that are available for Timed CSP.

In forming a semantics for Timed CSP it is clear that we have to attach times to the events that happen, since otherwise we would not be making distinctions that we evidently want to make. We assume the *No Zeno* principle that no process can perform infinitely many events (whether visible or invisible) in a finite time, but do allow a sequence of events all to happen at the same time. The main operator that constrains available models is hiding: in $P \setminus X$ no X event can be offered by P for more than zero time, for otherwise the process with X hidden would violate the principle of maximal process. It turns out that this means that we need to know what a timed process refuses at every moment of time, with the refusal at the time t of an event corresponding to what is refused instantaneously after the last event at t.

The combination of the principle of maximal progress and the need to make models compositional under the CSP hiding operator (which turns visible actions into τs that are forced before time passes) makes the range of models for Timed CSP more restricted than for untimed. It is necessary to record the set of events refused at every point in a behaviour where time advances. Divergence is a much reduced issue, since thanks to the no-Zeno assumption any divergence

is necessarily spread over infinite time – which when we are modelling time simplifies things greatly. In fact divergence will not be considered in the models we use in this paper.

In the case of continuous time this means that we have to record refusals as a subset of $\Sigma \times \mathbb{R}^+$ to accompany traces which attach a time in $\mathbb{R}^+$ (the non-negative real numbers) to each event, where the times increase, not necessarily strictly, through the trace. In fact, timed refusals are unions of sets of the form $X \times [t_1, t_2)$ where $0 \leq t_1 < t_2 < \infty$ –*refusal tokens.* $[t_1, t_2)$ is a *half-open interval* that contains t_1, all x with $t_1 < x < t_2$ but not t_2. This corresponds to the idea that if an event happens *at* time t then the refusal recorded at that time is the set of events refused at the same time *after* the event. So in $a \rightarrow P$, there will be behaviours in which a occurs at time 1, all events other than a are refused in the interval $[0, 1)$ and, on the assumption that the event a takes time δ to complete, all events including a are refused in the interval $[1, 1 + \delta)$.

So the *Timed Failures model* ($\mathcal{F}_T$) representation of a process consists of pairs of the form $(t, \aleph)$ (*timed failures*), where t us such a timed trace, and $\aleph$ is such a timed refusal. First introduced in [8], There have been a number of variants of this model over the years. The author analysed these in [14] and in this paper adopts the same version of $\mathcal{F}_T$, namely one

- Where causality is permitted within a single instant: for example one can have the timed trace $\langle (a, 1), (b, 1) \rangle$ but not the timed trace $\langle (b, 1), (a, 1) \rangle$.
- Where timed traces (as recorded) are finite (i.e. have only finitely many timed events) but where timed refusals can extend through all time, though they are finitary in the sense that they are the union of a countable set of refusal tokens $X \times [t_1, t_2)$ where the number of t_1s less than any fixed $t \in \mathbb{R}^+$ is finite.
- Where unfolding recursion takes no time, but only *time guarded* recursions in which no recursive call can be made before some $\delta > 0$ are permitted. The latter is to ensure that no process has a Zeno behaviour in which infinitely many actions can occur in a finite time.

Timed failures can be extended, if we wish, by an analogue of the divergence information used in $\mathcal{N}$. However, rather than record that a particular timed failure $(s, \aleph)$ is divergent (i.e. is accompanied by an infinite series of τs) we record the smallest time after the end of s (or ∞ if there is none) after which, when observing $(s, \aleph)$, we can be sure it must have become stable (i.e. no further τ will be enabled if no further visible event occurs. So the *Timed Failures Stability Model* $\mathcal{FS}_T$ represents a process as a set of triples $(s, \aleph, t)$ where $(s, \aleph)$ is one of its timed failures and t is the unique stability time associated with this.

Just as it is easy to extract the untimed traces of a process by deleting the times from timed traces, it is possible to extract natural values in the untimed models $\mathcal{F}$ and $\mathcal{N}$ from a process's representation in $\mathcal{FS}_T$. Untimed divergences come from timed triples $(s, \emptyset, \infty)$ by deleting the times in s to get *untime*(s). Stable failures from triples $(s, \aleph, t)$ with $t < \infty$: the latter gives the untimed failure whose trace is *untime*(s) and which refuses $\{a \mid \exists\, t'.(a, t') \in \aleph \wedge t' \geq t\}$.

In other words, anything that is refused after the time when the process must have become stable becomes part of the untimed refusal set.

So, for any Timed CSP process, we have a way of constructing values in each of the canonical untimed models. In each case this will refine the value you could have calculated by mapping the syntax into untimed CSP (i.e. mapping each $\mathit{WAIT}\, t$ component to SKIP). This substantiates the statement that the timed semantics is consistent with the untimed one, but the refinement might well be strict since modelling at the timed level can give us certainty about how nondeterminism in the untimed model will be resolved.

For example, consider $((a \rightarrow P) \,\square\, (\mathit{WAIT}\, 1;\ a \rightarrow Q)) \setminus \{a\}$ (where $\mathit{WAIT}\, 1$ could be replaced by any process whose untimed semantics has just the traces $\{\langle\rangle, \langle\checkmark\rangle\}$ and which always terminates on the empty trace after a non-zero time). The untimed semantics will identify this with $P \setminus \{a\} \sqcap Q \setminus \{a\}$ since they cannot tell that the τ resulting from the hiding of the left-hand a will always happen at time 0, with the a resolving the choice and excluding Q from doing anything before it starts. So in this case we get proper refinement. While the above approach is arguably the most natural way of linking timed and untimed theories from the perspective of the untimed theory, it misses out on two important things from the point of view of the timed models.

- Firstly, it ignores the most natural Timed CSP model $\mathcal{F}_T$: divergence and stability play no essential role in the semantics of Timed CSP as they do in the untimed version.
- Secondly, it ignores the fact that we can easily observe permanent refusal of a set of events X without stability: if no member of X is accepted in the states that appear between an infinite series of τ events, necessarily taking an infinite time, then we can reasonably equate this with untimed (i.e. permanent) refusal.

The natural way of extracting failures from timed failures, recalling that our version of $\mathcal{F}_T$ permits refusals that extend over an infinite period, is to map $(s, \aleph)$ to (s', X), where X is the largest X such that there is $t \geq \mathit{end}(s)$ with $X \times [t, \infty) \subseteq \aleph$. We will call this function from $\mathcal{F}_T$ to sets of failures Ψ. This gets the $\mathcal{F}_T$ value of STOP just right, though applying it to the same value with $\mu\, p. \mathit{WAIT}\, 1;\ p$ in mind gives (of course) the untimed semantics of STOP which is nothing like that in any standard untimed model of $\mu\, p. \mathit{WAIT}\, 1;\ p$.

What we have to accept is that, for processes that untimed CSP regards as divergent, the above mapping calculates something that is different from – and for some purposes superior to – the results calculated directly in the untimed models. After all $\mu\, p.\mathit{SKIP};\ p$ (the untimed analogue of $\mu\, p. \mathit{WAIT}\, 1;\ p$) will, from the perspective of the external user, sit there failing to accept any communication offered to it, just like STOP.

The range of Ψ is precisely the sets of failures that can occur in $\mathcal{N}$ and smaller than the set of those that occur in $\mathcal{F}$ because every untimed trace s' of the underlying process P has $(s', \emptyset)$ in $\Psi(P)$: if s is any timed trace that maps to s' then certainly $(s, \emptyset) = (s, \emptyset \times [\mathit{end}(s), \infty))$ is in P. In other words, Ψ maps $\mathcal{F}_T$ to the sets of failures of divergence-free processes.

Timewise refinement, as defined by Schneider, is a relation between untimed specifications *Spec*, generally expressed as divergence-free CSP processes and certainly members of the range of Ψ, and Timed CSP processes P:

$$Spec_{SF} \sqsubseteq_{TF} P \quad \Leftrightarrow \quad Spec \sqsubseteq \Psi(P)$$

where $\sqsubseteq$ is reverse containment over sets of failures.

We will see some examples to illustrate this definition later in this paper.

3 Digitisation and discrete time

FDR, as documented in [10, 12, 1] is a model checker for untimed CSP, whose algorithms manipulate discrete representations of discrete state machines.The key to verifying Timed CSP on it has been the theory of digitisation, in which the notation is re-interpreted over a discrete time domain (the natural numbers $\mathbb{N}$) and results proved to establish links between the semantics of a process over the two domains.

This restricts attention to *integer* Timed CSP, where all $WAIT\ t$ processes have $t \in \mathbb{N}$, with these the only delays introduced by a process as opposed to its environment. In the continuous semantics of this language, events can still happen at any time in $\mathbb{R}^+$, but in the discrete semantics they only happen at members of $\mathbb{N}$. Corresponding to $\mathcal{F}_T$ there is a *discrete timed failures model* $\mathcal{F}_{DT}$ in which there is a single refusal set following the zero or more events that happen at each integer time. There are a number of possible representations of this model, but we follow [14]: the extra event *tock* (which allows us to count the present time, and has no analogue in the continuous model) represents the regular passage of time, with all events between each pair of *tock*s being considered to happen at the same time as the preceding *tock*. (Events preceding the first *tock* happen at time 0.) There is a refusal set before each *tock* in each such trace. Traces are infinite but contain only finitely many non-*tock* events.

The theory of *digitisation* was introduced by Henzinger, Manna and Pnueli in [4] as a way of proving properties about continuous systems (specifically, timed automata) by analysing discrete approximations. It was adapted for Timed CSP by Ouaknine [6, 7] who showed that one can prove certain properties of systems over the continuous model $\mathcal{F}_T$ by demonstrating analogous properties of the same process's discrete semantics over $\mathcal{F}_{\mathbf{DT}}$. In particular he showed that every integer Timed CSP program has the property of being *closed under digitisation*, meaning that if $(s, \aleph)$ is in its $\mathcal{F}_T$ representation, then so is $[(s, \aleph)]_\epsilon$ for each $0 < \epsilon \leq 1$: this transforms each event and end-point of a refusal token $X \times [t_1, t2)$ to itself is an integer, and to one of the two surrounding integers otherwise:

- t_1 is $\lfloor t \rfloor$, where $\lfloor t \rfloor$ is the largest integer no greater than t.
- For $\epsilon < 1$, $t_\epsilon = \lfloor t \rfloor$ for if $frac(t) < \epsilon$ and $\lceil t \rceil$ otherwise, where $frac(t) = t - \lfloor t \rfloor$ and $\lceil t \rceil$ is the smallest integer no less than t.

Such integer behaviours map naturally to those recorded in $\mathcal{F}_{\mathbf{DT}}$ and are in fact members of the process's semantics in that model. It follows that if,

whenever the continuous time semantics of a process have a timed failure $(s, \aleph)$ that violates some specification S, there is some ϵ such that $[(s, \aleph)]_\epsilon$ also fails it, then we can determine whether an integer Timed CSP process satisfies S by considering only the discrete semantics.

A good example is provided by timewise refinement: if P fails to be a timewise refinement of the untimed specification S, this can only be because $\Psi(P)$ contains a failure not in S, or in other words there is a timed failure of the form $(s, X \times [t, \infty))$ for some s, $t \geq \mathit{end}(s)$ and X such that $(\mathit{untime}(s), X) \notin S$.

The digitisation property set out above implies that, in fact for *any* ϵ, $[(s, X \times [t, \infty)]_\epsilon$ is an integer behaviour that Ψ maps to $(\mathit{untime}(s), X)$. It follows that $S_{SF} \sqsubseteq_{TF} P$ if and only the same thing holds when judged over the discrete time semantics.

As set out in [6, 12], the discrete time semantics for Timed CSP can be calculated by systematically translating the Timed CSP language to *tock*-CSP, namely the usual CSP language with the addition of the special event *tock* representing the regular passage of time[4]. This translation has been automated in both FDR2 [1] and prototypes of FDR3. The essence of this that any process syntax contained within a `Timed(et){...}` section is automatically translated into the corresponding *tock*-CSP processes, using the *event timer* `et`, namely a mapping from events to the integer times taken to complete them.

The maximal progress property of Timed CSP is implemented by prioritising internal τ actions over *tock*. In other words, *tock* actions can only occur from stable states of the standard CSP operational semantics of the *tock*-CSP process.

This uses the priority operator $\mathbf{Pri}_{\leq}(P)$ specified in [13, 12], and now implemented as `prioritise(P,As)` in which `P` is a process and `As` a list of disjoint sets of events. `head(As)` consists of the events whose priority is equivalent to τ, which successive sets having lower priority. The operational semantics is that if the priority of event x is higher than that of event y, then y cannot occur from a state where P has x available. Events not in the union of `As` have no place in the priority order: they neither prevent and nor are they prevented by others. In the "blackboard" version $\mathbf{Pri}_{\leq}(P)$, $\leq$ represents a partial order on $\Sigma \cup \{\tau\}$ with some restrictions on the position of τ that are discussed in [13], and which are automatically satisfied by this machine readable version.

4 Slow abstraction

The idea of slow abstraction was introduced in [13]. $\mathcal{S}_A(P)$ represents how P appears to a user who can see the complement of A, on the assumption that events in that set are controlled by a user who habitually delays them, but not permanently. Thus the process is not prevented in making its offers outside A, but equally it is never permanently blocked by the abstracted user.

To consider $\mathcal{S}_A(P)$ we assume (as with lazy abstraction) that P is divergence-free. It differs from $P \setminus A$ in that as well as recording the refusals P makes when

[4] In fact this translation is to an extended version of *tock*-CSP since some Timed CSP operators need new untimed operators as their analogues.

it can refuse the whole of A (so $P \setminus A$ is stable), we also look at the series of sets it can refuse prior to it accepting each member of A in an infinite sequence of these. If all of these refuse a set X, then P will obviously not accept any member of X along the sequence.

This makes sense in the context of *slow* abstraction because we can suppose that all but finitely many A events happen from stable states, the abstracted user having waited for the divergence-free process to complete all its urgent τs before performing the next member of A.

Because such an *unstable refusal* can only happen at the end of an observed behaviour, and necessarily each finite trace of $P \setminus A$ is followed either a stable or unstable refusal in $\mathcal{S}_A(P)$, we identify it with a set of failures which always corresponds to the failures of a member of the failures divergence model $\mathcal{N}$.

Slow abstraction was introduced and studied in [13], and in particular a technique was introduced for deciding using FDR whether a failures specification *Spec* is refined by $\mathcal{S}_A(P)$. When an unstable failure of the latter violates *Spec*, this necessarily takes an infinite number of steps, and so the only way that FDR can detect this is as a divergence. This means that $\mathcal{S}_A(P)$ cannot be realised directly in the language of FDR, but rather we have to check the refinement somewhat obliquely. We will introduce it below for the case relevant to the present paper, namely when $A = \{tock\}$ and P is the *tock*-CSP translation of an integer Timed CSP implementation under the timed-priority model which ensures that all *tock*s happen from stable states.

For such a process, $P \setminus \{tock\}$ is never stable since no Timed CSP process ever refuses *tock*. In fact it should not be too hard to see that $\mathcal{S}_{\{tock\}}(P)$ consists of exactly the failures $\Psi(P)$ used in determining timewise refinement, so in fact the statements $Spec \sqsubseteq \mathcal{S}_{\{tock\}}(P)$ and $Spec_{SF} \sqsubseteq_{TF} P$ are equivalent.

The method for deciding this is best introduced in two stages. Consider the process

$$(\mathbf{Pri}_{\leq}((P[\![^{tock,\, tock'}/tock,\, tock]\!])) \setminus \{tock\})$$

where $\leq$ prioritises every event other than the new event $tock'$ over $tock$; $tock'$ is incomparable with all. In this, the τs created by hiding *tock* can only happen from states of P in which only *tock* is possible. On the other hand, every state of P is reachable thanks to $tock'$. Therefore the above process can diverge if and only if P has a state which has an infinite behaviour consisting of a mixture of τs and *tock*s, the latter from states offering just *tock*.

This is exactly equivalent to P being a timewise refinement of the deadlock free specification

$$DF = \sqcap\{a \to DF \mid a \in \Sigma \setminus \{tock, tock'\}\}$$

We therefore know how to solve our problem for this specific *Spec*.

Following [13], we can generalise this to arbitrary *Spec* with the combination of the trace check $Spec \sqsubseteq_T P \setminus \{tock\}$ and checking the combination of P and a testing process *Test*(*Spec*) against the unstable deadlock specification above. *Test*(*Spec*) is constructed as follows.

If S is any process such that $\alpha S \subseteq \alpha \mathit{Spec}$ and $(\langle\rangle, \Sigma) \notin \mathit{failures}(S)$, we can define $NR(S)$ to be the set of those X that are subset minimal with respect to $(\langle\rangle, X) \notin \mathit{failures}(S)$. $NR(S)$ is nonempty because Σ is finite and $(\langle\rangle, \Sigma) \notin S$.

If S is a process that can deadlock immediately, let $NR(S) = \emptyset$.

Choose a new event d that is outside $\alpha \mathit{Spec} \cup \{\mathit{tock}, \mathit{tock}'\}$. (Note that $\alpha P \subseteq \alpha \mathit{Spec} \cup \{\mathit{tock}, \mathit{tock}'\}$ because we are assuming that $\mathit{Spec} \sqsubseteq_T P \setminus \{\mathit{tock}, \mathit{tock}'\}$.) For a set of refusals $R \neq \emptyset$, let

$$T(R) = \Box_{X \in R}\, d \to (?x : X \to DS)$$

and $T(\emptyset) = DS$, where $DS = d \to DS$. Note that $T(R) \underset{\alpha \mathit{Spec}}{\|} Q$, for Q a process such that $S \sqsubseteq_T Q$, can deadlock if and only if, when one of the sets $X \in R$ is offered to P when it has performed $\langle\rangle$, P refuses it. This parallel composition is therefore deadlock free if no member of R is an initial (stable) refusal of P. Now let

$$\mathit{Test}(S) = (?x : S^0 \to \mathit{Test}(S/\langle x\rangle)) \;\Box\; T(NR(S))$$

For Q such that $\mathit{Spec} \sqsubseteq_T Q$, the parallel composition $\mathit{Test}(\mathit{Spec}) \underset{\alpha \mathit{Spec}}{\|} Q$ is then deadlock free if and only if $\mathit{Spec} \sqsubseteq_F Q$, given that we know that $S \sqsubseteq_T P$: the composition can deadlock if and only if, after one of its traces s, P can refuse a set that S does not permit. In understanding this it is crucial to note that the ds of $T(NR(S))$, including the one in the initial state of $\mathit{Test}(S)$, can occur unfettered in the parallel composition because they are not synchronised with Q. The first d that occurs fixes the present trace as the one after which $\mathit{Test}(\mathit{Spec})$ checks to see that a disallowed refusal set (if any) does not appear in Q.

Our construction turns any case where Q fails Spec into a deadlock. It is very similar to the "watchdog" transformation for the usual failures model set out in [2]. The main difference is that ours is constructed with no τ actions: the visible action d replaces τ.

Consider the case where Q is replaced by P. This has the additional event tock which is not synchronised with $\mathit{Test}(\mathit{Spec})$, so the combination $\mathit{Test}(\mathit{Spec}) \underset{\alpha \mathit{Spec}}{\|} P$ can only deadlock in states where $P \setminus \{\mathit{tock}\}$ has a stable failure illegal for Spec. In fact this never happens for us because, as remarked above, $P \setminus \{\mathit{tock}\}$ is never stable.

We would like unstable failures of $\mathcal{S}_{\{\mathit{tock}\}}(P)$ to turn into unstable "deadlocks", namely unstable refusals of Σ, in $\mathcal{S}_{\{\mathit{tock}\}}(\mathit{Test}(\mathit{Spec}) \underset{\alpha \mathit{Spec}}{\|} P)$. This is confirmed by the following result, the proof of which is essentially the same as that about general $\mathcal{S}_A(P)$ in [13], when we take into account that in our case $P \setminus \{\mathit{tock}\}$ is never stable.

Theorem 1. *Under our assumptions, including* $\mathit{Spec} \sqsubseteq_T P \setminus \{\mathit{tock}\}$*,* $\mathcal{S}_{\{\mathit{tock}\}}(P)$ *has an unstable failure that violates Spec if and only if* $\mathcal{S}_{\{\mathit{tock}\}}(\mathit{Test}(\mathit{Spec}) \underset{\alpha \mathit{Spec}}{\|}$

P) has an unstable failure of the form (s, Σ). Furthermore $\mathcal{S}_{\{tock\}}(P) \sqsupseteq_F Spec$ if and only if

$$(\mathbf{Pri}_{\leq}((Test(Spec) \underset{\alpha Spec}{\|} (P[\![^{tock,\,tock'}/_{tock,\,tock}]\!]))) \setminus \{tock\}$$

is divergence-free.

When combined with our earlier observation that, thanks to a digitisation argument, proving timewise refinement of a *Spec* for the *tock*-CSP translation P' of an integer Timed CSP process P also proves for P, the above result gives us a way of deciding questions of timewise refinement on FDR.

5 Examples

The above methods have proved both effective and efficient in the examples we have experimented with to date. A variety of CSP files is included with version of this paper posted on the author's web site. These include Timed CSP versions of the Alternating Bit and Sliding Window Protocols, which are naturally shown to give timewise refinements of the untimed buffer specification. These files use the same versions of these protocols used in example files published with [1], but whereas the specifications proved of the original versions were inevitably parameterised by timing details, the new versions show how to establish untimed correctness in a form requiring no timing details.

This has the efficiency advantage that there is no need to experiment with different timing parameters when verifying a timed system, and in many cases the absence of timing in the specification reduces the overall number of states visited for a given implementation P. However the need to use divergence checking places extra demands on FDR which are always likely to impose moderate restrictions on the size and speed of the check.[5]

Consider the following machine-readable Timed CSP description of a token ring:

```
datatype Packet = Full.(Nodes,Nodes,Data) | Empty

channel ring:Nodes.Packet
channel input,output:Nodes.Nodes.Data
succ(n) = (n+1)%N
```

[5] At the time of writing (May 2013) the author has the choice of using FDR2 or a prototype FDR3. In FDR2 the data structures used make relatively small (say 8M or less checks) typically almost as fast as checks for finitary properties, but slow down substantially beyond that. FDR3's checking for divergence is more efficient than FDR2's and less limited, but presently operates essentially sequentially, whereas traces and stable failures checking use parallelism across multi-core devices. This situation is likely to improve, but parallel checking of divergence properties is always likely to be more difficult and less efficient than the parallelism available for checking finitary properties.

```
allone(X) = 1

Timed(allone){
Token(k,Empty) = input.k?dest?x -> Token(k,Full.(k,dest,x))
                 [] (WAIT(1);ring.succ(k).Empty -> NoToken(k))

Token(k,Full.(f,t,x)) = if t==k then
                       output.k.f.x -> ring.succ(k).Empty -> NoToken(k)
                       else ring.succ(k).Full.(f,t,x) -> NoToken(k)

NoToken(k) = ring.k?p -> Token(k,p)

Alpha(k) = {|ring.k,ring.succ(k),input.k,output.k|}

init(k) = if k <  Tokens then Token(k,Empty) else NoToken(k)

RING = (|| k:Nodes @[Alpha(k)]init(k))\{|ring|}
}
```

This sets up an `N`-place circular ring with `Tokens` tokens that rotate around it, carrying messages between the nodes. We might ask the question of whether it acts as a buffer between each ordered pair of nodes. An untimed specification which expresses this for fixed nodes `i` and `j` is

```
Buff(i,j,<>) = input.i?k?x -> if j==k then Buff(i,j,<x>)
                                    else Buff(i,j,<>)

Buff(i,j,xs^<x>) = output.j.i.x -> Buff(i,j,xs)
                [] if #xs == Tokens-1 then STOP
                 else (STOP |~|
                    input.i?k?y -> if j==k then Buff(i,j,<y>^xs^<x>)
                                     else Buff(i,j,xs^x>))
```

This has as its alphabet all inputs at node `i` and the outputs at `j` that come from `i`.

The right way to judge our ring against this is to hide all other outputs and lazily abstract all other inputs. This corresponds to the assumption that users accept all outputs from the ring eagerly, and can arbitrarily decide what messages other than those considered by the specification.

The main complication in checking timewise refinement they way we are advocating is the need to need to transform the untimed specification into the corresponding testing process. In our case this is

```
Test(i,j,<>) = [] x:Data @ d -> input.i.j.x -> DS
               [] (|~| k:Nodes @ if k==j then
                            input.i.j?x -> Test(i,j,<x>)
                       else input.i.k?x -> Test(i,j,<>))
```

```
Test(i,j,xs^<y>) = d -> output.j.i.y -> DS
                   [] output.j.i.y -> Test(i,j,xs)
                   [] #xs < Tokens-1 & (|~| k:Nodes @ if k==j then
                                  input.i.j?x -> Test(i,j,<x>^xs^<y>)
                              else input.i.k?x -> Test(i,j,xs^<y>))
```

Fortunately this process is automatable and could be implemented as primitive in a future version of FDR.

The ring fails this specification because the other nodes might repeatedly fill every token (in the lazy abstraction) before it reaches node `i`. In other words it fails the aspect of the failures specification that the ring, when there is no message in transport from `i` to `j`, it must eventually accept an input at `i`. The error shows up as a divergence in FDR, whose debugger decomposes this into the `Test` process constantly offering node `i` an input, and each of a cycle of states of the token ring refusing that input.

This can be repaired by adjusting the tokens that pass around so that upon being emptied they do not immediately gain the ability to transport another message. Rather, successive tokens gain this ability when they reach one of the nodes, where the node with this property rotates. Thus no node can indefinitely be denied the ability to accept a message from its user.

The versions of the ring with and without this added control can be found amongst the files available with this paper. The version with does satisfy the requirement under timewise refinement.

The modified ring, in common with the original one set out above, only makes input offers to each user for a single unit of time each time an empty token passes. This illustrates the fact that satisfying a specification formulated in terms of timewise refinement does not imply that offers the specification requires must be made permanently beyond some point; rather that they must be made at an infinite number of times beyond some point. The implementation is allowed to make progress (in our case via the tokens circulating) and changing the set of events offered.

6 Conclusions

We have given a practical and theoretically sound way of checking timewise refinement in FDR. It proved to be straightforward to link the usual relation between specifications and continuous Timed CSP, to a tractable relation between the same specification and the *tock*-CSP process corresponding to the continuous one.

Testing this requires both a renaming and hiding trick which converts illegal unstable failures into divergences, and the conversion of the specification into a suitable watchdog process.

We have demonstrated that this approach works for a range of examples, and hope that others find it useful. To enable this it would be helpful if functionality were added to FDR to create a *Test*(*Spec*) process automatically from each *Spec*

References

1. P. Armstrong, G. Lowe, J. Ouaknine, and A.W. Roscoe, *Model checking Timed CSP*, To appear in Proceedings of HOWARD, Easychair.
2. M.H. Goldsmith, N. Moffat, A.W. Roscoe, T. Whitworth and M.I. Zakiuddin, Watchdog transformations for property-oriented model-checking, FME 2003: Formal Methods, LNCS 2805, 2003.
3. C.L. Heitmeyer, B.G. Labaw and R.D. Jeffords, *A benchmark for comparing different approaches for specifying and verifying real-time systems*, DTIC, 1993.
4. T.A. Henzinger, Z. Manna, and A. Pnueli, *What good are digital clocks? In Proceedings of the Nineteenth International* Colloquium on Automata, Languages, and Programming (ICALP 92), volume 623, pages 545-558. Springer LNCS, 1992.
5. C.A.R. Hoare, *Communicating sequential processes*, Prentice Hall, 1985.
6. J. Ouaknine, *Discrete analysis of continuous behaviour in real-time concurrent systems*, Oxford University D.Phil thesis, 2001.
7. J. Ouaknine, *Digitisation and full abstraction for dense-time model checking*, TACAS Springer LNCS, 2002.
8. G.M. Reed and A.W. Roscoe, *A timed model for communicating sequential processes*, Theoretical Computer Science **58**, 249-261, 1988.
9. G.M. Reed and A.W. Roscoe, *The timed failures-stability model for CSP*, Theoretical Computer Science **211**, 85-127, 1999.
10. A.W. Roscoe, *Model checking CSP*, in 'A classical mind: essays in honour of C.A.R. Hoare', Prentice Hall, 1994.
11. A.W. Roscoe, *The theory and practice of concurrency* Prentice Hall, 1997.
12. A.W. Roscoe, *Understanding concurrent systems*, Springer, 2010.
13. A.W. Roscoe and P.J. Hopcroft, *Slow abstraction through priority*, To appear in Festschrift proceedings for He Jifeng, ICTAC 2013.
14. A.W. Roscoe, and Huang Jian *Checking noninterference in Timed CSP*, FAC, **25**, pp 1-33, 2013.
15. S.A. Schneider, *Timewise refinement for communicating processes*, Science of Computer Programming, **28**, pp 43–90, 1997.
16. S.A. Schneider, *Concurrent and real-time systems: the CSP approach*, Wiley, 2000.

Runtime Quantitative Verification: Applications and Research Challenges

Radu Calinescu, Simos Gerasimou, Kenneth Johnson and Yasmin Rafiq

Department of Computer Science, University of York
Deramore Lane, Heslington, York YO10 5GH, United Kingdom
{radu.calinescu,sg778,kenneth.johnson,yr534}@york.ac.uk

Abstract

A growing number of software, embedded and cyber-physical systems are expected to adapt continually to changes in the environments they operate in. Many of these systems are deployed in safety-critical and business-critical applications from domains including healthcare, finance and defence, and must comply with strict dependability and performance requirements. To achieve such compliance, formal techniques traditionally employed in the design of critical systems must also be used during their operation. This paper describes how a formal technique termed *quantitative verification* can be exploited in a runtime setting. We present several successful runtime applications of the technique, and discuss the research challenges that must be addressed in order to extend its applicability to other runtime scenarios.

Contents

1 Introduction **1**

2 Runtime Quantitative Verification **3**

3 Applications **4**
3.1 Dynamic Power Management 4
3.2 Self-Adaptive Service-Based Systems 5
3.3 Continual Verification of Cloud-Deployed Services 7

4 Towards Efficient Runtime Quantitative Verification **8**
4.1 Compositional Quantitative Verification 8
4.2 Incremental Quantitative Verification 9
4.3 Parametric Quantitative Verification 10

5 Other Research Challenges **11**

6 Conclusion **11**

1 Introduction

Ensuring that today's software, embedded and cyber-physical systems comply with their requirements is a great challenge. Traditional verification approaches are often ineffective for these systems, which are increasingly used in critical applications and required to evolve continually in response to changes in their environment, objectives and internal state [6, 28]. Recent research advocated the use of runtime variants of established verification techniques to address this challenge, leading to the development of techniques such as *runtime verification* [2, 3, 44, 45, 48],

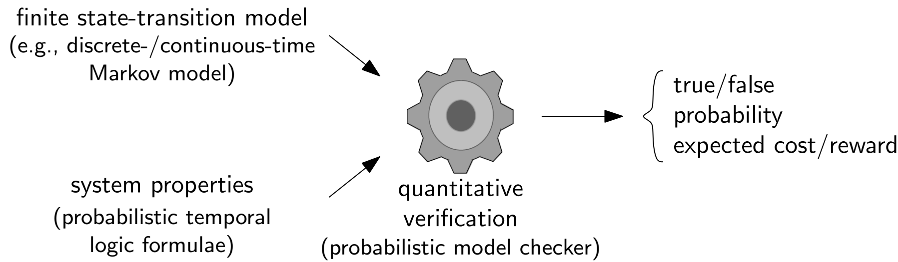

Figure 1: Quantitative verification

runtime certification [50] and *runtime quantitative verification* [8, 15, 17, 20, 22, 23]. This paper discusses the advantages, applications and current limitations of the last of these techniques.

Quantitative verification is a mathematically based technique for analysing the correctness, performance and reliability of systems that exhibit stochastic behaviour [25, 36, 39, 40]. The technique operates with finite state-transition models comprising states that correspond to different configurations of a real-world system, and edges associated with the possible transitions between these states. Depending on the purpose of the analysis, the edges are annotated with transition probabilities or transition rates. The types of models with probability-annotated transitions include discrete-time Markov chains and Markov decision processes, while the edges of continuous-time Markov chains are annotated with transition rates. Given one of these models and a system property specified formally in temporal logic extended with probabilities and costs/rewards, the technique analyses the model exhaustively in order to evaluate the property. Examples of properties that can be established using the technique include the probability that a fault occurs within a specified time period, the expected response time of a service under a given workload, and the expected energy usage of a disk drive. Quantitative verification is typically performed entirely automatically by tools termed *probabilistic model checkers*, as illustrated in Figure 1. Widely used probabilistic model checkers include PRISM [41], MRMC [37] and Ymer [52].

Runtime quantitative verification was introduced in [17, 20] and further refined in [8, 15, 16, 22, 23]. The technique and its typical use within the monitor-analyse-plan-execute (MAPE) autonomic computing control loop [38] are described in Section 2. We then summarise several successful applications of runtime quantitative applications in Section 3. These applications come from our own work and that of other research groups, and span domains including dynamic power management, self-adaptive service-based systems and analysis of cloud-deployed service reliability. In many of these scenarios, runtime quantitative verification is integrated into small or medium-scale self-adaptive systems. The high computation overhead and large memory footprint of the technique need to be addressed before this success can be reproduced for large-scale systems. This great challenge and preliminary work to identify ways of overcoming it are detailed in Section 4, and several other research challenges are summarised in Section 5. We conclude the paper with a brief analysis that summarises our view on the role of runtime quantitative verification in the development, management and operation of self-adaptive software, embedded and cyber-physical systems.

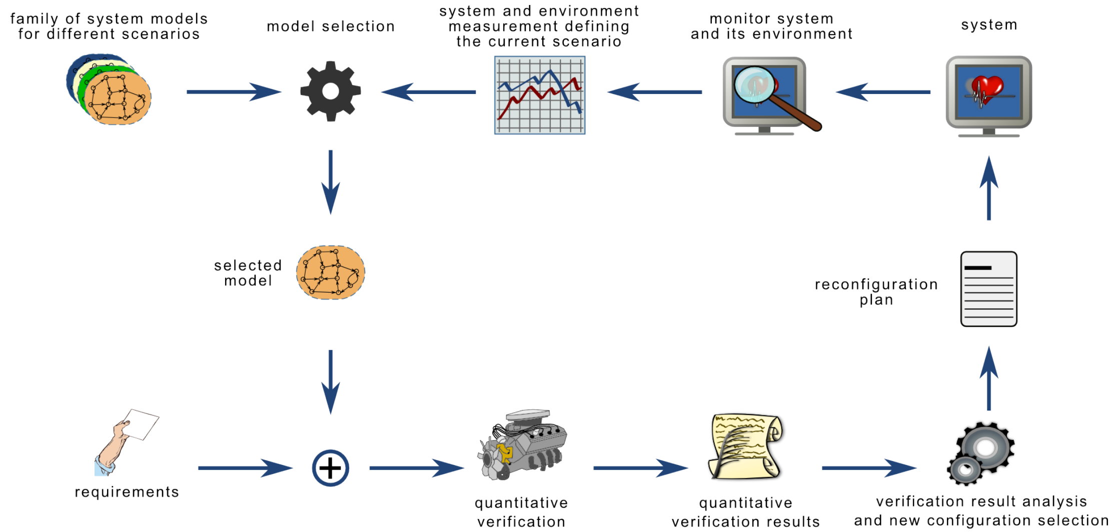

Figure 2: Using runtime quantitative verification to implement the MAPE control loop

2 Runtime Quantitative Verification

Like most formal verification techniques, quantitative verification is traditionally used in off-line settings, e.g., to evaluate the performance-cost tradeoffs of alternative system designs, and to establish if existing systems comply with their reliability requirements. In the latter case, systems in violation of their requirements undergo off-line maintenance, and are eventually replaced with suitably modified system versions. This approach does not meet the demands of emerging application scenarios in which systems need to be continually verified as they adapt autonomously, as soon as a need for change is detected while they operate [5, 7].

Runtime quantitative verification addresses this need for continual verification of self-adaptive systems. As illustrated in Fig. 2, a self-adaptive system that uses the approach (and its environment) are monitored at runtime, and relevant changes are identified and quantified using fast on-line learning techniques. The resulting measurements enable the identification of the scenario that the system operates in, and the selection of a suitable model from a family of system models associated with different such scenarios. As an example, Bayesian learning techniques are used in [10, 18, 20] to monitor the changing probabilities of successful service invocation for a service-based system, and thus to determine the transition probabilities of a parameterised Markovian chain that models the system.

The model selected through the monitoring process described above is then analysed using quantitative verification, to identify and/or predict violations of quality-of-service (QoS) requirements such as response time, availability and cost. When requirement violations are identified or predicted, the results of the analysis support the synthesis of a reconfiguration plan. This plan comprises adaptation steps whose execution ensures that the system will continue to satisfy its requirements despite the changes identified in the monitoring step.

Using quantitative verification (or other *formal methods* [15, 13]) to implement the MAPE control loop provides irrefutable guarantees about the compliance of the selected reconfiguration plans with the considered system requirements. This key advantage has led to the successful exploitation of runtime quantivative verification in a wide range of self-adaptive systems, several of which are described in the next section.

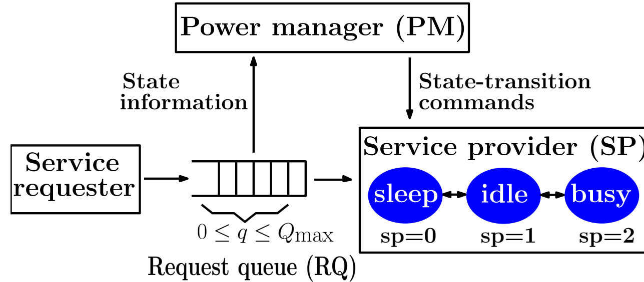

Figure 3: Architecture of a DPM-enabled disk drive

3 Applications

This part of the paper presents applications that illustrate the use of runtime quantitative verification in three different domains. In the interest of readability and conciseness, all applications are presented at a high level, using a minimum of mathematical notation. For the readers interested in the full technical details of any of these applications, we include references to the work where each application was originally reported.

3.1 Dynamic Power Management

In this application, runtime quantitative verification is used to support the dynamic power management of a disk drive with the architecture shown in Fig. 3. The device consists of a *service provider* that handles requests generated by a *service requester* and stored in a *request queue*. The service provider has several possible states corresponding to different power consumption and service rates, and its state transitions are controlled by a *power manager* that aims to optimise power consumption while maintaining an acceptable level of service for the device. In our work from [17], we implemented this power manager using the runtime quantitative verification approach described in Section 2. This was achieved using a parameterised continuous-time Markov chain (CTMC) model of a Fujitsu disk drive with the structure in Fig. 3, and which was originally proposed in [49].

The CTMC model used for the application is parameterised by the request inter-arrival rate and by the "switch-to-sleep" probability with which the power manager switches the disk drive into a low-power "sleep" state when the request queue is empty. The changing value of the former parameter is learnt through monitoring the length of the request queue, and runtime quantitative verification is used to continually select the "switch-to-sleep" probability that maximises a multi-objective *utility function* [4, 51]. This utility function has the generic form $\sum_{i=1}^{n} w_i \mathit{utility}_i$, where the weights $w_i > 0$, $1 \leq i \leq n$, encode the trade-offs among $n > 0$ system objectives, and each objective function $\mathit{utility}_i$, $1 \leq i \leq n$, is an analytical expression of the system parameters. As shown in Fig. 4a, the utility function for the disk drive is obtained through combining two objective functions that partition the CTMC state space into optimal, suboptimal and "zero-utility" regions based on the length of the request queue and the power consumption associated with each state, respectively.

During the operation of the disk drive, changes in the monitored request inter-arrival rate trigger a runtime quantitative verification step. This step establishes the expected power consumption and request queue length associated with the new system state and a range of "switch-to-sleep" probabilities that can be used to configure the power manager. Fig. 4b shows how

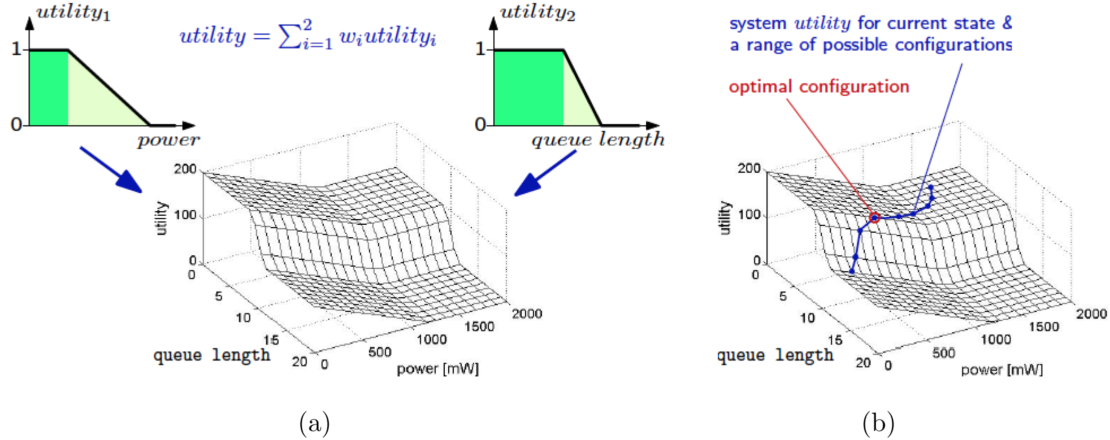

Figure 4: (a) The disk drive utility function integrates energy usage and performance-related objective functions that partition the CTMC state space into regions that are optimal (dark shaded), suboptimal (light shaded) and of zero utility. (b) Runtime quantitative verification enables the selection of the optimal configuration for the current state of the disk drive.

these possible configurations correspond to different system utility values, thus supporting the selection of a new optimal "switch-to-sleep" probability for the reconfiguration plan from Fig. 2.

3.2 Self-Adaptive Service-Based Systems

In this application, runtime quantitative verification is used to drive the dynamic selection of the components of service-based systems (SBSs). Built through the integration of third-party services deployed on remote datacentres and accessed over the Internet, SBSs are increasingly used in business- and safety-critical applications where they must comply with strict quality-of-service (QoS) requirements. Self-adaptive SBSs achieve this QoS compliance by selecting the *concrete services* for their operations dynamically, from sets of functionally equivalent third-party services with different levels of performance, reliability and cost. Fig. 5 depicts a self-adaptive SBS architecture in which this concrete service selection is underpinned by runtime quantitative verification.

Originally proposed in [9] and further extended in [8, 12, 11], the self-adaptive SBS architecture from Fig. 5 comprises $n \geq 1$ operations performed by remote third-party services. The $n \geq 1$ *intelligent proxies* in this architecture interface the SBS workflow with sets of remote service such that the i-th SBS operation can be carried out by $m_i \geq 1$ functionally equivalent services. The main role of the intelligent proxies is to ensure that each execution of an SBS operation is carried out through the invocation of a concrete service selected as described below. Whenever an instance of the i-th proxy is created, it is initialised with a sequence of "promised" service-level agreements $sla_{ij} = (p^0_{ij}, c_{ij}), 1 \leq j \leq m_i$, where $p^0_{ij} \in [0, 1]$ and $c_{i,j} > 0$ represent the provider-supplied probability of success and the cost for an invocation of service $s_{i,j}$, respectively.

The n proxies are also responsible for notifying a *model updater* about each service invocation and its outcome. The model updater starts from a developer-supplied initial Markovian model of the SBS workflow, and uses the online learning techniques from [10, 18] to adjust the transition probabilities of the model in line with these proxy notifications. Finally, the up-

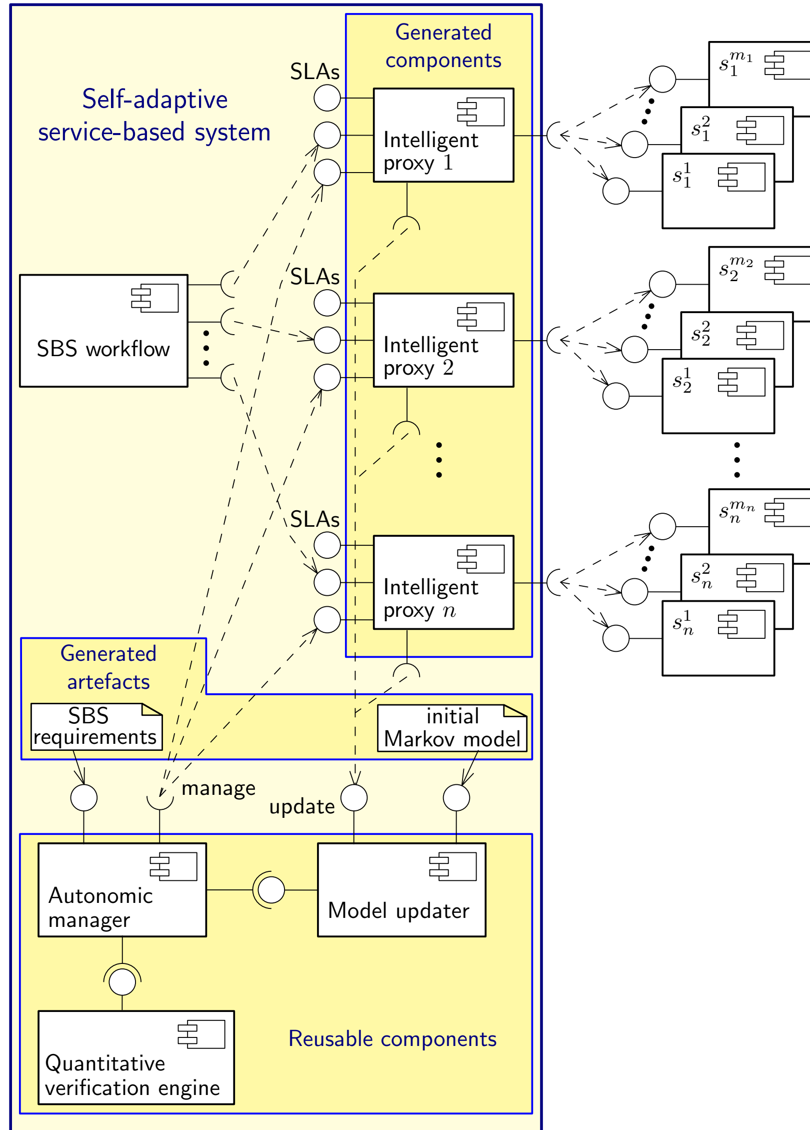

Figure 5: Self-adaptive service-based system whose dynamic service selection is driven by runtime quantitative verification, adapted from [12, 11].

to-date Markovian model maintained by the model updater is used by an *autonomic manager* that performs runtime quantitative verification to select the service combination used by the n proxies so that it satisfies the SBS requirements with minimal cost at all times. Accordingly, the proxies, model updater and autonomic manager with its quantitative verification engine implement the monitor-analyse-plan-execute (MAPE) autonomic computing loop from Fig. 2.

Our work from [10, 18] introduced a tool-supported SBS development framework in which many of the components and artefacts described above are either generated automatically (e.g., starting from an annotated UML activity diagram of the SBS workflow and from the WSDL

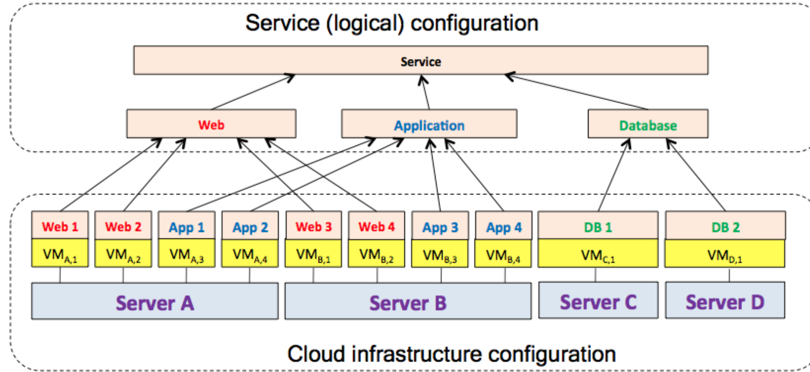

Figure 6: Three-tiered architecture of a cloud-deployed service

specifications of its concrete services) or workflow-independent and therefore reusable. The quantitative verification engine used by this framework is based on an embedded version of the PRISM probabilistic model checker [41], and the experiments in [10, 18] show that it can handle well the necessary "on the fly" analysis for a range of small-sized SBS workflows. Several approaches to reducing the overheads of runtime quantitative verification in order to extend its applicability to larger workflows are described in Section 4.

3.3 Continual Verification of Cloud-Deployed Services

The last application of runtime quantitative verification detailed in the paper was originally proposed in [14, 34], and involves the continual verification of the reliability of software services deployed on cloud computing infrastructure owned by the service provider. More specifically, we are interested in tracking the probability that a cloud-deployed *multi-tier software service* becomes unavailable as the cloud resources allocated to the service change over time. A multi-tier software service is a collection of interdependent components (or "functions"), each delivering a specific functionality of the service. Figure 6 depicts the deployment of a service whose architecture comprises three functions: web, application and database. Several instances of each function run on different virtual machines (VMs) that are deployed across four physical servers that are part of the cloud infrastructure.

Services deployed on cloud infrastructure take advantage of its elastic nature, scaling rapidly to meet demands by adding new servers, virtual machines and function instances. These types of changes are governed by policies and are often carried out programatically in response to fluctuations in the service workload. Other changes to the service happen unexpectedly. Examples of unexpected changes include hardware failure of physical servers or failure of one or more function instances arising from software faults. By applying quantitative verification at runtime, administrators of multi-tier services can gain valuable insight of the impact each change has on the service's reliability by obtaining precise answers to questions such as

Q1 What is the maximum probability of a service failing over a one-month time period?[1]

Q2 How will the probability of failure for my service be affected if one of its database instances is switched off to reflect a decrease in service workload?

[1] A service is deemed to have failed if all instances of one of its functions are unavailable.

Administrators (or automated scripts) then have the option of reacting with remedial action if the service fails to comply with its requirements as a result of an observed change (e.g., a failure of a hard disk on a physical server). They can also discard planned changes (e.g., a removal of a function instance) whose implementation would violate these requirements.

To achieve these objectives, our runtime quantitative verification approach from [14, 34] models the behaviour of the service components from Fig. 6 as probabilistic automata parameterised by the configuration variables and failure probabilities of these components. These parameters are updated to reflect the state of the cloud-deployed service after each observed or planned change. Safety properties encoded in probabilistic temporal logic and expressing the service reliability requirements are then reverified to quantify the impact of the change.

4 Towards Efficient Runtime Quantitative Verification

Formal verification is a well-established means for ascertaining the correctness, reliability and other QoS properties of software, embedded and cyber-physical systems. Nevertheless, existing formal verification techniques are notorious for their large computation overheads and memory footprints—a characteristic shared by quantitative verification. As an example, a complete model of the relatively simple cloud-deployed software service from Fig 6 was shown to comprise $176,381,406,182,650$ states and its properties could not be verified on a standard computer [14]. This *state explosion* [1] has a major impact when quantitative verification is used in an on-line setting, limiting the applicability of standard runtime quantitative verification to small-scale systems [28]. Overcoming this limitation represents the greatest challenge of runtime quantitative verification. In the remainder of this section, we present recent research that aims to address this challenge [14, 21, 22, 26, 31, 32, 34, 42, 43]. In doing so, we will organise the results of this research into three categories of approaches—incremental, compositional and parametric runtime quantitative verification.

4.1 Compositional Quantitative Verification

Many large-scale systems are assembled from $n > 1$ interdependent components. When this is the case, the formal verification of a large system involves the construction and analysis of a model of the form $M = M_1 \parallel M_2 \parallel \ldots \parallel M_n$, where M_i represents the model for the i-th system component. As verifying M as a monolithic model is associated with high overheads or may be intractable, *compositional verification* aims to establish system-level model properties from the properties derived through the analysis of its component-level models M_1 to M_n.

In its original form proposed in the seminal work of Pnueli [47], compositional verification involves establishing that the parallel composition of two models $M_1 \parallel M_2$ satisfies a global property $\mathcal{G}$ through verifying two premises independently. The first premise is that M_2 satisfies $\mathcal{G}$ when it is part of a system that satisfies an *assumption* (i.e., property) $\mathcal{A}$. The second premise is that $\mathcal{A}$ is satisfied by the remainder of the system (i.e., by M_1) under all circumstances. This can be expressed formally as a proof tree by using Pnueli's generalisation [47] of the Hoare triple notation [33]:

$$\frac{\langle true \rangle M_1 \langle \mathcal{A} \rangle,\ \langle \mathcal{A} \rangle M_2 \langle \mathcal{G} \rangle}{\langle true \rangle M_1 \parallel M_2 \langle \mathcal{G} \rangle}.$$

The technique is termed *assume-guarantee reasoning*, to distinguish it from other compositional verification approaches that have emerged more recently. Assume-guarantee reasoning has been

extended to probabilistic systems in [42], through the introduction of the *probabilistic assume-guarantee rule*

$$\frac{\langle true\rangle M_1\langle\mathcal{A}\rangle_{\geq p_1},\ \langle\mathcal{A}\rangle_{\geq p_1} M_2\langle\mathcal{G}\rangle_{\geq p_2}}{\langle true\rangle M_1 \parallel M_2\langle\mathcal{G}\rangle_{\geq p_2}},$$

where the models M_1 and M_2 are probabilistic automata, and the *probabilistic safety properties* $\langle\mathcal{A}\rangle_{\geq p_1}$, $\langle\mathcal{G}\rangle_{\geq p_2}$ encode the requirements that the "desirable outcomes" $\mathcal{A}$ and $\mathcal{G}$ are achieved with at least probabilities p_1 and p_2, respectively. The experiments carried out in [42] show that the approach can reduce the memory footprint and running time of quantitative verification by several orders of magnitude, and, in some cases, enable the verification of systems whose monolithic models could not fit in memory or whose verification did not terminate within 24 hours. Among other successful applications, the approach made possible the continual verification of cloud-deployed services described in Section 3.3.

4.2 Incremental Quantitative Verification

Changes occurring in a technical system are typically localised: they often affect only a small subset of system elements [28]. This creates the opportunity to speed up a sequence of runtime quantitative verifications of successive system configurations by reusing at least some verification results from one step of the sequence in the next step. Thus, incremental quantitative verification involves finding exactly what changed, and (re)verifying only the affected parts of the model, while exploiting in as much as possible the results of previous verification steps.

An approach that belongs to this category is proposed in [43], where the underlying graph of a Markov model is decomposed into Strongly Connected Components (SCCs), i.e., sets of states in which there is a path between any two states and which are maximal in the sense that no supersets are also strongly connected. This state partition permits a two-stage verification of the model in which each SCC is verified individually in the first stage, and the system-level verification involves integrating the SCC-level verification results in the second stage. When a change in state transition probabilities occurs, the technique reverifies only the SCCs affected by the change in its first stage, and reuses the previous verification results for the unaffected SSCs in its second stage.

The work from [43] is extended in [26], where the authors introduce incremental techniques for model construction and quantitative verification. Model construction is defined as the problem of synthesising a Markov decision process (MDP) from its description in a high-level modelling language. The incremental model construction technique in [26] identifies the states that need to be rebuilt after a change in the model description, reducing the set of high-level description statements that are evaluated in order to rebuild the updated MDP. For quantitative verification, [26] decomposes the system model into SCCs, as in [43]. However, [26] uses policy iteration to establish the correctness of a formula, while [43] applies value iteration. Experiments performed on a set of case studies showed significant improvement in computation time in both [43] and [26].

Finally, the approach introduced in [14, 34] extends the set of probabilistic assume-guarantee rules from [42] and uses the extended rule set to assemble *probabilistic assume-guarantee proof trees* for the verified safety properties. Changes in a component-based system are mapped to changes of individual nodes in this proof tree. As a result, the reverification of a safety property after a change can be performed incrementally, by analysing only the node directly affected by the change and, depending on the results of this analysis, its ancestor nodes affected by the new results. The approach was used successfully in the application described in Section 3.3, to reverify reliability properties of cloud-deployed services after individual component failures and

prior to carrying out system reconfigurations meant to reduce resource usage but which might violate safety system requirements [14, 34].

4.3 Parametric Quantitative Verification

Parametric quantitative verification techniques work by transforming system requirements into algebraic formulae which can then be evaluated efficiently at runtime. Daws' work on parametric model checking [19] introduces a new language-theoretic technique for the symbolic probabilistic model checking of a subset of probabilistic computation tree logic (PCTL) formulae over discrete-time Markov chains (DTMCs). The technique comprises three steps:

(i) the conversion of the analysed DTMC into a finite state automaton in which the transition probabilities are modelled as letters of an alphabet;

(ii) the synthesis of a regular expression that describes the language recognised by the automaton, through applying state elimination or inductive algorithms;

(iii) the recursive evaluation of the regular expression, yielding an exact rational value when the original transition probabilities are rational or a rational function when the DTMC is parameterised.

Runtime quantitative verification can be sped up by using this technique off-line, to reduce the requirements of a system to algebraic formulae that are then evaluated efficiently at runtime, when the actual values of their parameters are obtained through monitoring the system. This section presents two types of approaches based on this principle—state elimination algorithms [31, 32] and linear algebra approaches [21, 22, 24].

Elimination Algorithms for Parametric Verification. Hahn *et al.* [32], drawing upon the results from [19], devise an effective algorithm for computing reachability properties in parametric Markov models. The novelty compared to [19] resides in intertwining the state elimination with an early evaluation of the target rational function. In particular, in each iteration after the state elimination step, transitions are labelled directly with the rational function instead of regular expression. This allows the on-the-fly simplification of the rational function in each iteration, by taking advantage of cancellations, symmetries and simplifications of arithmetic expressions and by replacing numerical expressions with their values.

In their later work from [31], Hahn *et al.* consider the problem of parameter synthesis for the PCTL formulae associated with parametric models. The aim is to synthesise sets of parameter values for which a given PCTL formula holds. The approach applies state-space exploration techniques that partition the parameter space into rectangular regions with the property that either the PCTL formula holds for all parameter values in the region or it does not. The size of these regions is decreased over a number of recursive iterations, although the approach can leave a limited area of the state space undecided in the interest of efficiency.

Linear Algebra Approaches for Parametric Verification. In [22], Filieri *et al.* propose a two-stage approach for efficient runtime probabilistic model checking that exploits the results from [19, 32]. The two stages of the approach, termed *pre-computation* and *verification*, are executed at design time and runtime, respectively. At design time, the approach starts from a DTMC model of a system, a set of reliability-related system requirements, and a set of variables representing unknown DTMC transition probabilities whose values can be appraised only at runtime. Given this information, the pre-computation stage translates the system

requirements into equivalent algebraic expressions parameterised by the variables associated with the unknown transition probabilities. These expressions are then evaluated efficiently in the runtime verification stage, by replacing the variables with their actual values obtained while monitoring the system. The approach is extended to DTMCs augmented with reward structures in [21, 24].

5 Other Research Challenges

With a number of efficient techniques for runtime quantitative verification starting to emerge as explained in the previous section, several other challenges will also need to be addressed to enable the adoption of the approach in mainstream engineering practice. First, practitioners should be relieved from the error-prone task of having to manually assemble the parametric system models underpinning runtime quantitative verification. These parametric models should be synthesised automatically from available data such as application logs and runtime observations of the system, or from models that practitioners are familiar with (e.g., UML models). Preliminary research to automate the model synthesis by using the two types of solutions has been reported in [29, 35, 46] and [11, 27], respectively.

Another area requiring significant work is the continual updating of the parametric system models, e.g., through fixing the values of their parameters based on runtime observations. The positive results reported, for instance, in [10, 12, 18, 20, 27] need to be extended and integrated into reusable components that practitioners can use in their applications.

Finally, practitioners should also be offered tools that automate the translation of system requirements into the temporal logic formulae used by runtime quantitative verification. Performing this translation manually is known to be error prone, and requires formal verification expertise that is not currently common in industry. The extensive study in [30] and the successful adoption of its results in [9] confirm that the process can be automated when the original system requirements are expressed in a domain-specific language that practitioners are familiar with.

6 Conclusion

We overviewed recent research on runtime quantitative verification, a technique used to establish the correctness of reconfiguration actions undertaken by self-adaptive systems. The technique has been adopted successfully in multiple application domains, and we described several applications from these domains in order to illustrate its role within the closed control loop of self-adaptive systems.

So far, most self-adaptive systems whose reconfiguration is driven by runtime quantitative verification are small or medium-scale. Extending the applicability of the technique to large-scale systems is hindered by its high computation overheads and large memory footprint. Devising the much lighter-weight variants of runtime quantitative verification that are required for this extension represents a great challenge. The compositional, incremental and parametric quantitative verification techniques proposed in recent years and summarised in Section 4 represent promising approaches to addressing this challenge. Nevertheless, significant research is still needed to evaluate the effectiveness of these new techniques in real-world applications, to identify their benefits and address their limitations, and to provide tools that practitioners who are not experts in formal verification can use to exploit the approach.

Acknowledgement

This work was partly supported by the UK Engineering and Physical Sciences Research Council grant EP/H042644/1.

References

[1] C. Baier and J.-P. Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[2] H. Barringer and K. Havelund. A Scala DSL for trace analysis. In M. Butler and W. Schulte, editors, *FM 2011: Formal Methods*, volume 6664 of *Lecture Notes in Computer Science*, pages 57–72. Springer Berlin / Heidelberg, 2011.

[3] H. Barringer, K. Havelund, D. Rydeheard, and A. Groce. Rule systems for runtime verification: A short tutorial. In S. Bensalem and D. Peled, editors, *Runtime Verification*, volume 5779 of *Lecture Notes in Computer Science*, pages 1–24. Springer Berlin / Heidelberg, 2009.

[4] R. Calinescu. Reconfigurable service-oriented architecture for autonomic computing. *International Journal On Advances in Intelligent Systems*, 2(1):38–57, June 2009.

[5] R. Calinescu. When the requirements for adaptation and high integrity meet. In *Proceedings of the 8th workshop on Assurances for self-adaptive systems*, ASAS '11, pages 1–4, New York, NY, USA, 2011. ACM.

[6] R. Calinescu. Emerging techniques for the engineering of self-adaptive high-integrity software. In J. Camara et al., editors, *Assurances for Self-Adaptive Systems*, volume 7740 of *LNCS*, pages 297–310. Springer, 2013.

[7] R. Calinescu. Emerging techniques for the engineering of self-adaptive high-integrity software. In J. Camara, R. Lemos, C. Ghezzi, and A. Lopes, editors, *Assurances for Self-Adaptive Systems*, volume 7740 of *Lecture Notes in Computer Science*, pages 297–310. Springer Berlin Heidelberg, 2013.

[8] R. Calinescu, C. Ghezzi, M. Kwiatkowska, and R. Mirandola. Self-adaptive software needs quantitative verification at runtime. *Communications of the ACM*, 55(9):69–77, September 2012.

[9] R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli. Dynamic QoS management and optimization in service-based systems. *IEEE Trans. Softw. Eng.*, 37:387–409, 2011.

[10] R. Calinescu, K. Johnson, and Y. Rafiq. Using observation ageing to improve Markovian model learning in QoS engineering. In *2nd ACM/SPEC Intl. Conf. on Performance Engineering*, pages 505–510, 2011.

[11] R. Calinescu, K. Johnson, and Y. Rafiq. Developing self-verifying service-based systems. In *28th Intl. IEEE/ACM Conference on Automated software Engineering*, 2013. To appear.

[12] R. Calinescu, K. Johnson, and Y. Rafiq. Using continual verification to automate service selection in service-based systems. Technical Report YCS-2013-484, Dept. of Computer Science, University of York, 2013. `http://www.cs.york.ac.uk/ftpdir/reports/2013/YCS/482/YCS-2013-484.pdf`.

[13] R. Calinescu and S. Kikuchi. Formal methods @ runtime. In *Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems*, volume 6662 of *Lecture Notes in Computer Science*, pages 122–135. Springer, 2011.

[14] R. Calinescu, S. Kikuchi, and K. Johnson. Compositional reverification of probabilistic safety properties for large-scale complex IT systems. In *Large-Scale Complex IT Systems*, volume 7539 of *LNCS*, pages 303–329. Springer, 2012.

[15] R. Calinescu, S. Kikuchi, and M. Kwiatkowska. Formal methods for the development and verification of autonomic IT systems. In P. Cong-Vinh, editor, *Formal and Practical Aspects of Autonomic*

Computing and Networking: Specification, Development and Verification, pages 1–37. IGI Global, 2010.

[16] R. Calinescu and M. Kwiatkowska. CADS*: Computer-aided development of self-* systems. In M. Chechik and M. Wirsing, editors, *Fundamental Approaches to Software Engineering (FASE 2009)*, volume 5503 of *Lecture Notes in Computer Science*, pages 421–424. Springer, March 2009.

[17] R. Calinescu and M. Z. Kwiatkowska. Using quantitative analysis to implement autonomic IT systems. In *Proceedings of the 31st International Conference on Software Engineering, ICSE 2009*, pages 100–110. IEEE Computer Society, 2009.

[18] R. Calinescu and Y. Rafiq. Using intelligent proxies to develop self-adaptive service-based systems. In *7th Intl. Symp. on Theoretical Aspects of Software Engineering*, pages 131–134, 2013.

[19] C. Daws. Symbolic and parametric model checking of discrete-time markov chains. In Z. Liu and K. Araki, editors, *Theoretical Aspects of Computing - ICTAC 2004*, volume 3407 of *Lecture Notes in Computer Science*, pages 280–294. Springer Berlin Heidelberg, 2005.

[20] I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburrelli. Model evolution by run-time adaptation. In *Proc. 31st Intl. Conf. Software Engineering (ICSE'09)*, pages 111–121, 2009.

[21] A. Filieri and C. Ghezzi. Further steps towards efficient runtime verification: Handling probabilistic cost models. In *Software Engineering: Rigorous and Agile Approaches (FormSERA), 2012 Formal Methods in*, pages 2–8, 2012.

[22] A. Filieri, C. Ghezzi, and G. Tamburrelli. Run-time efficient probabilistic model checking. In *Proceedings of the 33rd International Conference on Software Engineering*, pages 341–350. IEEE Computer Society, 2011.

[23] A. Filieri, C. Ghezzi, and G. Tamburrelli. A formal approach to adaptive software: continuous assurance of non-functional requirements. *Formal Asp. Comput.*, 24(2):163–186, 2012.

[24] A. Filieri and G. Tamburrelli. Probabilistic verification at runtime for self-adaptive systems. In J. Camara, R. Lemos, C. Ghezzi, and A. Lopes, editors, *Assurances for Self-Adaptive Systems*, volume 7740 of *Lecture Notes in Computer Science*, pages 30–59. Springer Berlin Heidelberg, 2013.

[25] V. Forejt, M. Kwiatkowska, G. Norman, and D. Parker. Automated verification techniques for probabilistic systems. In M. Bernardo and V. Issarny, editors, *Formal Methods for Eternal Networked Software Systems (SFM'11)*, volume 6659 of *LNCS*, pages 53–113. Springer, 2011.

[26] V. Forejt, M. Kwiatkowska, D. Parker, H. Qu, and M. Ujma. Incremental runtime verification of probabilistic systems. In S. Qadeer and S. Tasiran, editors, *Runtime Verification*, volume 7687 of *Lecture Notes in Computer Science*, pages 314–319. Springer Berlin Heidelberg, 2013.

[27] S. Gallotti, C. Ghezzi, R. Mirandola, and G. Tamburrelli. Quality prediction of service compositions through probabilistic model checking. In *Proceedings of the 4th International Conference on Quality of Software-Architectures: Models and Architectures*, QoSA '08, pages 119–134, Berlin, Heidelberg, 2008. Springer-Verlag.

[28] C. Ghezzi. Evolution, adaptation, and the quest for incrementality. In *Large-Scale Complex IT Systems*, volume 7539 of *LNCS*, pages 369–379. Springer, 2012.

[29] C. Ghezzi, M. Pezzè, and G. Tamburrelli. Adaptive rest applications via model inference and probabilistic model checking. In F. D. Turck, Y. Diao, C. S. Hong, D. Medhi, and R. Sadre, editors, *IM*, pages 1376–1382. IEEE, 2013.

[30] L. Grunske. Specification patterns for probabilistic quality properties. In Robby, editor, *30th International Conference on Software Engineering (ICSE 2008)*, pages 31–40. ACM, 2008.

[31] E. M. Hahn, T. Han, and L. Zhang. Synthesis for pctl in parametric markov decision processes. In *Proceedings of the Third international conference on NASA Formal methods*, NFM'11, pages 146–161, Berlin, Heidelberg, 2011. Springer-Verlag.

[32] E. M. Hahn, H. Hermanns, and L. Zhang. Probabilistic reachability for parametric markov models. In *Proceedings of the 16th International SPIN Workshop on Model Checking Software*, pages 88–106, Berlin, Heidelberg, 2009. Springer-Verlag.

[33] C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, Oct. 1969.

[34] K. Johnson, R. Calinescu, and S. Kikuchi. An incremental verification framework for component-based software systems. In *Proc. 16th Intl. ACM Sigsoft Symposium on Component-Based Software Engineering*, pages 33–42, 2013.

[35] K. Johnson, S. Reed, and R. Calinescu. Specification and quantitative analysis of probabilistic cloud deployment patterns. In K. Eder, J. Lourenço, and O. Shehory, editors, *Haifa Verification Conference*, volume 7261 of *Lecture Notes in Computer Science*, pages 145–159. Springer, 2011.

[36] J.-P. Katoen. Model checking meets probability: A gentle introduction. In *Engineering Dependable Software Systems*, NATO Science for Peace and Security Series - D, pages 1–29. IOS Press, 2013.

[37] J.-P. Katoen, M. Khattri, and I. S. Zapreev. A Markov reward model checker. In *Quantitative Evaluation of Systems*, pages 243–244, Los Alamitos, 2005. IEEE Computer Society.

[38] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *IEEE Computer Journal*, 36(1):41–50, January 2003.

[39] M. Kwiatkowska. Quantitative verification: Models, techniques and tools. In *Proc. 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pages 449–458. ACM Press, September 2007.

[40] M. Kwiatkowska, G. Norman, and D. Parker. Stochastic model checking. In M. Bernardo and J. Hillston, editors, *Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07)*, volume 4486 of *LNCS (Tutorial Volume)*, pages 220–270. Springer, 2007.

[41] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV'11*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

[42] M. Kwiatkowska, G. Norman, D. Parker, and H. Qu. Assume-guarantee verification for probabilistic systems. In J. Esparza and R. Majumdar, editors, *Proc. 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'10)*, volume 6105 of *LNCS*, pages 23–37. Springer, 2010.

[43] M. Kwiatkowska, D. Parker, and H. Qu. Incremental quantitative verification for Markov decision processes. In *Dependable Systems Networks (DSN), 2011 IEEE/IFIP 41st International Conference on*, pages 359–370, 2011.

[44] M. Kyas, C. Prisacariu, and G. Schneider. Run-time monitoring of electronic contracts. In *Proceedings 6th International Symposium on Automated Technology forVerification and Analysis (ATVA'08)*, 2008.

[45] M. Leucker and C. Schallhart. A brief account of runtime verification. *J. Log. Algebr. Program.*, 78(5):293–303, 2009.

[46] D. Perez-Palacin, R. Calinescu, and J. Merseguer. log2cloud: log-based prediction of cost-performance trade-offs for cloud deployments. In S. Y. Shin and J. C. Maldonado, editors, *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 397–404. ACM, 2013.

[47] A. Pnueli. In transition from global to modular temporal reasoning about programs. In K. R. Apt, editor, *Logics and models of concurrent systems*, pages 123–144. Springer-Verlag New York, Inc., New York, NY, USA, 1985.

[48] A. Pnueli and A. Zaks. PSL model checking and run-time verification via testers. In *Proceedings 14th International Symposium on Formal Methods (FM'06)*, pages 573–586, 2006.

[49] Q. Qiu, Q. Wu, and M. Pedram. Stochastic modeling of a power-managed system: construction and optimization. In *Proc. International Symposium on Low Power Electronics and Design*, pages 194–199. ACM Press, 1999.

[50] J. M. Rushby. Runtime certification. In *Proceedings 8th International Workshop on Runtime Verification (RV'08)*, pages 21–35, 2008.

[51] W. E. Walsh, G. Tesauro, J. O. Kephart, and R. Das. Utility functions in autonomic systems. In *Proceedings of the 1st International Conference on Autonomic Computing*, pages 70–77, New York, USA, 2004.

[52] H. L. S. Younes. Ymer: A statistical model checker. In K. Etessami et al., editors, *Computer Aided Verification*, volume 3576 of *LNCS*, pages 429–433. Springer, 2005.

Towards autonomous embedded systems

Sagar Behere, Martin Törngren, Jad El-khoury and DeJiu Chen

Kungliga Tekniska Högskolan, Stockholm, Sweden
behere@kth.se, martint@kth.se, jad@kth.se, chendj@kth.se

Abstract

Machines incorporating embedded systems display a trend towards increasing autonomy. In this position paper, we outline an approach for introducing autonomy in embedded system architectures. The approach involves the creation of an artificial consciousness within the machine. We propose that the artificial consciousness may be represented in the form of domain specific reference architectures. We illustrate the approach with the aid of a validated reference architecture for cooperative driving.

1 Introduction

Two trends are apparent in the design and construction of many machines that surround us:

1. There is an increasing usage of electronics, computers and software within the machines (a.k.a embedded systems)
2. There is an increasing desire to make the machines autonomous, where autonomy is defined as 'operation without direct human intervention'.

Therefore, a question that will gain increasing importance is:

What should the hardware and software architecture of a machine look like, so that autonomous operation is easy to achieve?

This work proposes an approach for answering the above question. The approach is valid under a specific set of pre-conditions and constraints, which are also described in this work.

The proposed approach is still a work in progress. Nevertheless, an application of the approach in the form of an automotive reference architecture for cooperative driving[3, 4] and two instantiations thereof has already been made. This text describes the approach, the reference architecture and its instantiations as well a non-exhaustive list of questions whose solutions need to be found, to develop the approach further.

2 Proposed approach

2.1 Autonomy, intelligence and machine consciousness

One of the principal aids to achieving machine autonomy is machine intelligence. Intelligence can be defined as the ability of a system to act appropriately in an uncertain environment, where appropriate action is that which increases the probability of success, and success is the achievement of behavioral sub-goals that support the system's ultimate goal[1]. There may be several ways to construct a machine so that it *displays* intelligence. The display of intelligence can be evaluated solely by observing external behavior, without concerning oneself with the mechanisms within the machine that generate the displayed intelligent behavior (If it looks,

walks and quacks like a duck...). If the external behavior of the machine is indistinguishable from that which would be generated by an intelligent entity, then the machine may be considered as intelligent[7] (a.k.a the 'Turing test'). The external display of intelligence must, however, be differentiated from the internal mechanisms in the machine that give rise to that behavior.

For the purpose of this text, we constrain[1] the term 'consciousness' as the quality or state of being aware of something within oneself. This leads to the question, "Who, or What, is it that is being aware?" This question must be answered. We will answer it in section 2.3.

It must be pointed out that consciousness does not imply intelligence and intelligence does not imply autonomy. The converse is also true: intelligence does not imply consciousness, nor does autonomy imply intelligence.

We claim that **the combination of consciousness and intelligence is a useful engineering method to achieve machine autonomy, when the machine is a system composed out of multiple sub-systems**. In the context of such a machine, our notion of consciousness can be reified by a distinct sub-system that

1. is aware of the overall purpose(s) of the machine, can understand the short term goals of the machine's user and the behavior expected for the fulfillment of those goals
2. recognizes the presence of the other sub-systems of the machine, and knows how they should interact to generate expected machine behavior
3. understands the environment that the machine operates in, and the expected interactions of the machine with its environment
4. is capable of interpreting the commands of the machine's user and orchestrating behavior of other sub-systems in order to execute those commands

Such a consciousness subsystem may contain algorithms for machine learning and intelligence, which could be combined with elements of intelligence present in other sub-systems. With such a construction, the machine may be deemed to be equipped with mechanisms for understanding the commands of its user, and for executing those commands. This is nothing but the essence of machine autonomy.

2.2 Pre-conditions and constraints

Our approach to machine autonomy assumes the existence of the following pre-conditions and constraints

- Embedded computer systems and software are the primary means to generate and control machine behavior
- The embedded systems are the sole focus area for the achievement of machine autonomy. All efforts to realize machine autonomy will be concentrated on the embedded hardware and software only.
- The embedded systems within a machine are organized into sub-systems. Each sub-system consists of one (or a small group of) computers.

[1] The terms 'conscious' and 'consciousness' have been expounded with significantly greater meaning elsewhere. See for example Van Gulick [8]

- There is constant communication between the embedded sub-systems. This communication may be used for exchanging data and/or altering the specific software functions being executed by a sub-system.

- There exist legacies of proven sub-system designs, together with strong reasons for minimizing changes to these sub-system designs.

2.3 The Self and progressive autonomy

Our approach to embedded systems autonomy consists of introducing into the system architecture a sub-system that reifies the notion of machine consciousness. We denote such a sub-system by the term 'Self'. It is this Self that lends an identity to the machine. Users interacting with the machine are in fact interacting with the Self. Earlier in section 2.1 we asked, "Who, or What is it that is aware within the machine?" The answer is: the Self. In this way, our approach endeavors to partially mimic the notion of consciousness and self-awareness in human beings.

From the architectural perspective, some interesting questions are:

- What should be the structure and interfaces of the Self?

- What are the patterns of interaction between the Self and the other sub-systems?

- How should the sub-systems be designed so that they can interact more easily with the Self?

- What particular sequence of design iterations should be followed to evolve existing architectures towards those that incorporate and utilize the Self?

- How are cross-cutting extra-functional properties like system safety, reliability, error management etc. affected by the Self? Can the presence of the Self be exploited to favorably affect these properties?

Introduction of the Self into the architecture needs to be complemented by examination of aspects related to formal representations of system construction and desired behavior. Formal representations provide the Self with the knowledge necessary for appropriate reasoning and control of the system. The representations in turn will be affected by the algorithms used by the Self for reasoning and decision making. Given the dependency of these aspects on the domain, task and implementation specific details, it might not be possible to specify a sufficiently general solution that works for all types of embedded systems. General solutions however, could be in the form of *domain specific reference architectures* that are instantiable for specific use cases. Reference architectures[5] are essentially proven solution templates and patterns that are useful for solving a specific category of problems. An example of a reference architecture based on our approach is given in section 3.

One particular salient benefit of our approach must be highlighted. The benefit is that the approach enables progressive autonomy. Progressive autonomy means that the autonomy of the system is gradually increased over successive design iterations. This implies that the degree of human intervention needed to operate the machine decreases over successive product versions. Progressive autonomy is important for two reasons

1. It enables cautious increase in capabilities of a function that is inherently susceptible to uncertainty and errors that have safety consequences.

2. Existing and legacy systems can be the starting point. These can be gradually evolved towards autonomy, making large design changes unnecessary. This is appreciated by commercial companies whose products are already in the market (example: automotive manufacturers).

The reason why progressive autonomy is enabled by our approach is that the capabilities of the Self (together with the architectural changes necessary to take advantage of those capabilities) can be developed in an incremental fashion. At its simplest, the Self need be no more than a passive component that is fed some status data by the rest of the subsystems. It need take no active role in determining and affecting the system behavior, but could be used, for example, to provide diagnostic information and/or warnings. Next, the Self may be allowed to interpret user inputs as intentions to achieve specific system behaviors, while still permitting the Self no control over the other sub-systems. The inputs and internal reasoning performed by the Self could be logged over time to validate the correctness of the related algorithms, and only then could the Self be granted executive powers that affect the functioning of the system.

3 Application example: A reference architecture for cooperative driving

This section gives an example of adding a Self as an additional sub-system to a set of existing sub-systems. The example also demonstrates a recursive characteristic of the approach: the Self is implemented as an additional sub-system which in turn consists of sub-subsystems. One of the sub-subsystems is a Self (sub-Self?)!

3.1 A Self for cooperative driving

The electrical/electronic (E/E) sub-systems of a modern automobile meet the constraints listed in section 2.2. For the purpose of introducing autonomy, an automobile can be considered as a set of interconnected, embedded computer (sub-)systems, each of which has a specific purpose. Our approach to autonomy suggests the addition of another sub-system (the Self) that can function as the system's consciousness. To illustrate this approach, we considered the specific problem of creating autonomous motion under cooperative driving conditions. Cooperative driving conditions are those where continuous wireless communication exists between a vehicle and its surroundings, which consist of the local road infrastructure as well as the other vehicles in the vicinity. The Self of the autonomous system should then understand that

1. the purpose of the system is autonomous driving (under cooperative driving conditions) and the short term goals of the system are to navigate the vehicle in a specific environment

2. there are other sub-systems in the vehicle (like the engine, the brakes and the transmission) which have defined roles and that the correct interaction of these sub-systems will generate the desired behavior

3. the environment of the vehicle consists of objects that include other vehicles and road infrastructure (traffic lights, speed limit signs etc.) and how the vehicle should react to the presence/absence of these objects

When given the appropriate commands by the user, the cooperative driving Self should be able to correctly interpret the commands and orchestrate the other sub-systems to realize safe,

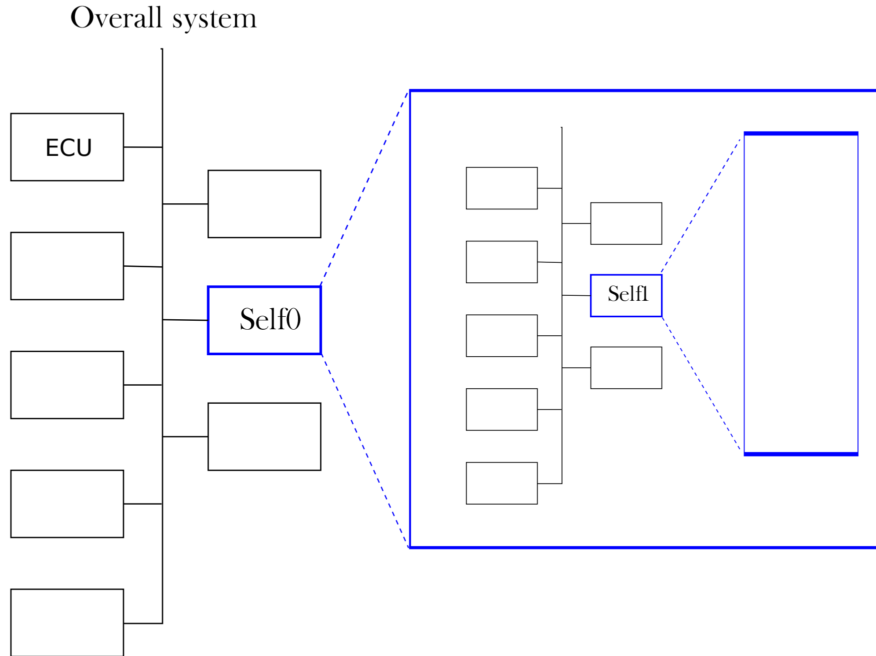

Figure 1: Recursive Selves

cooperative driving. This problem is domain specific and sufficiently detailed to generate a reference architecture for the Self, as mentioned in section 2.3.

Accordingly, a reference architecture for cooperative driving was created, which is described in [4]. This reference architecture was instantiated[6, 2] on two separate occasions, one of which was the Grand Cooperative Driving Challenge (GCDC) 2011. The GCDC consisted of vehicle platooning on public roads. An instantiation of the reference architecture was installed on a Scania R730 commercial truck, and the modified truck successfully completed the driving challenge. A second instantiation of the reference architecture was installed on a Scania R480 commercial truck, which was then utilized during further cooperative driving demonstrations. The two instantiations differed in capabilities and had very little in common.

Thus, on two separate occasions, the concept of adding a Self i.e. consciousness sub-system (for the purpose of autonomous cooperative driving) was demonstrated to produce desired system behavior.

3.2 A Self within a Self

The introduction of a Self happens in the form of an extra sub-system in the system architecture (see Figure 1). If we denote this extra sub-system as 'Self0', then it is entirely possible that Self0 itself comprises of multiple sub-subsystems and that one of these sub-subsystems is a Self (denoted 'Self1' in Figure 1) and so on. Thus, the approach demonstrates recursive characteristics.

In the case of our particular reference architecture for cooperative driving, one of the key architectural elements is a Self, present in the form of a Supervisor component. Specifically, *"..It is the supervisor that encodes an understanding of the various architectural elements, their*

capabilities and limitations. Thus, it is the supervisor that is aware of the presence of the world model, the control and other elements[of the reference architecture] *and how they must function in order to generate specific behaviors of the cooperative driving system. The supervisor "knows" what behavior is expected of the cooperative driving system in a given context and uses them to achieve the expected behavior. The elements in turn pass on all unknown inputs, locally unresolvable errors and requests to the supervisor and expect instructions on how they should proceed."*[4].

Thus the reference architecture provides a blueprint for a Self (Self0, as per Figure 1) that generates autonomous behavior in the vehicle. Within the reference architecture, there is another Self (Self1, as per Figure 1) that generates the desired behavior of the reference architecture. This "second-level Self" illustrates precisely the same principles of creating autonomous embedded systems as outlined by our proposed approach.

4 Conclusions and future work

We have outlined an approach towards embedded systems architecture to achieve machine autonomy. Parts of the approach have been validated in the context of cooperative driving for which a reference architecture was developed.

Further work is required to elaborate the approach and to encompass other autonomy settings. Introducing autonomy will also require specific efforts for addressing safety and reliability aspects. In particular there is a dichotomy between the determinism required by safety practices vs. the dynamic behavior which is inherent in autonomy. Safety standards, legislation and supporting technology all need further work

References

[1] J.S. Albus. Outline for a theory of intelligence. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(3):473–509, 1991. ISSN 00189472. doi: 10.1109/21.97471. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=97471`.

[2] Sagar Behere. Scoop Technical Report: Year 2011. Technical report, KTH Royal Institute of Technology, Stockholm, 2011. URL `http://kth.diva-portal.org/smash/get/diva2:567028/FULLTEXT01`.

[3] Sagar Behere. *Architecting Autonomous Automotive Systems*. Licentiate thesis, KTH, Stockholm, 2013. URL `http://kth.diva-portal.org/smash/record.jsf?searchId=6&pid=diva2:615888`.

[4] Sagar Behere, Martin Törngren, and Dejiu Chen. A reference architecture for cooperative driving. *Journal of Systems Architecture*, 2013.

[5] Philippe Kruchten. *The Rational Unified Process*. Rational Software White Paper. Addison-Wesley, 2003. ISBN 0321197704.

[6] Jonas Må rtensson, Assad Alam, and Sagar Behere. The Development of a Cooperative Heavy-Duty Vehicle for the GCDC 2011: Team Scoop. *IEEE Transactions on Intelligent Transportation Systems*, 13(3):1033–1049, September 2012. ISSN 1524-9050. doi: 10.1109/TITS.2012.2204876. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6236179http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6236179`.

[7] A. M. Turing. Computing Machinery and Intelligence. *Mind*, LIX(236):433–460, 1950. ISSN 0026-4423. doi: 10.1093/mind/LIX.236.433. URL http://mind.oxfordjournals.org/cgi/doi/10.1093/mind/LIX.236.433.

[8] Robert Van Gulick. Consciousness. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. 2011. URL http://plato.stanford.edu/archives/sum2011/entries/consciousness/.

Synchronous Specialization of Alf for Cyber-Physical Systems

Alessandro Gerlinger Romero[1], Klaus Schneider[2] and Maurício Gonçalves Vieira Ferreira[3]

[1] Brazilian National Institute for Space Research, São Paulo, Brazil.
romgerale@yahoo.com.br
[2] Department of Computer Science, University of Kaiserslautern, Germany.
schneider@cs.uni-kl.de
[3] Brazilian National Institute for Space Research, São Paulo, Brazil.
mauricio@ccs.inpe.br

Abstract

Systems engineers often use SysML as a vendor-independent language to model cyber-physical systems. However, SysML does not provide an executable form of behaviors which is needed for simulation to detect critical issues as soon as possible. In this paper, we therefore present an action language for foundational UML (Alf) specialization that introduces the synchronous-reactive model of computation to SysML. This is done by definition of not explicitly constrained semantics of timing, concurrency, and inter-object communication. The smart parking system, a well-known cyber-physical system benchmark, was selected to evaluate this specialization. Our initial results show that the proposed specialization does not add complexity to the task of modeling using SysML, and leads to concise and precise behavioral definitions.

1 Introduction

Cyber-physical systems (CPSs) are obtained by integration of computational and physical processes: Embedded computers and networks monitor and control physical processes with feedback loops where physical processes affect computations and vice versa [17]. According to Cartwright et. al. [9], the difficulty in modeling CPSs comes from the diversity of these systems. Therefore, the most promising approach to mitigate this problem is to develop expressive and precise modeling languages.

As a result, a large number of languages and formalisms have been proposed to model CPSs [8]. One particular branch of these languages follows the synchronous-reactive model of computation (MoC) [4], which has several advantages for specifying, modeling, and verifying of reactive real-time systems [17].

The synchronous-reactive MoC provides a precise behavioral model using discrete reaction steps as the fundamental model of time, while computation and communication are executed in zero-time, and parallel composition is defined as a conjunction of behaviors [4]. There is a solid mathematical foundation that supports synchronous-reactive MoCs, which allows in particular formal analysis and verification. Languages based on this MoC, like Esterel [6], have therefore been developed and used for safety-critical systems [4, 28]. Some of them, like Quartz [29], have been extended for CPSs [3] by special kinds of variables and statements to deal with continuous time.

Comparing an implementation based on the synchronous-reactive MoC with an alternative asynchronous implementation for a dual redundant flight guidance system, Miller et. al. [20] made the following observation: *"the properties themselves are more difficult to state, were*

weaker than could be achieved in the synchronous case, and required considerable complexity to be added to the model to ensure that even the weakened properties were true".

Meanwhile, the Object Management Group (OMG) and the International Council on Systems Engineering (INCOSE) are developing the Systems Modeling Language (SysML) [25, 31]; a general-purpose modeling language for systems engineering applications based on the Unified Modeling Language (UML) [23]. SysML has demonstrated a capability for top-down design refinement, but the lack of formal foundations of SysML results in imprecise behavioral models.

In this paper, we present a specialization to the action language for foundational UML (Alf) [26] for behavioral modeling of CPSs. The hypothesis of this work is that a specialization of Alf according to the synchronous-reactive MoC can be sufficiently expressive to model the discrete behavior of CPSs using SysML. Consequently, adhering to the synchronous-reactive MoC, we will benefit from a solid mathematical foundation [4, 6, 29].

The remainder of this paper is organized as follows: in Section 2, related works are explored briefly; in Section 3, we present the initial approach; in Section 4, a case study is presented; in Section 5, we briefly discuss the initial approach and the case study; finally, conclusions are shared in the final section.

2 Related Work

There is a large number of research papers about formalizing the semantics of models using UML, and consequently, also about SysML. Hußmann [16] proposes the following classification for approaches concerning structural semantics: (a) naive set-theory, (b) meta-modeling, and (c) translation. This classification can be used for the works focused on behavioral semantics.

For example, Graves and Bijan [15] propose an approach where behaviors defined using SysML state machine diagrams are axiomatized using set/type theory. Alf [26], and the foundational subset for executable UML models (fUML) [24], follow the meta-modeling approach because the semantics of behaviors is described operationally using fUML itself. The circularity is broken by the base semantics of fUML which is specified using first order logic. However, Benyahia et. al. [5] show that fUML, and also Alf, are not directly feasible to safety-critical systems because the MoC defined in the fUML execution model (as it is) is nondeterministic and sequential.

Following (c), i.e., translation, Bousse et. al. [7] define a method to transform a subset of SysML in B method representations; the selected subset of SysML covers behavioral definitions expressed by Alf. Later, the B method representation is proved by a specialized tool. Abdelhalim et. al. [1] define a method that receiving state machine diagrams and activity diagrams (according to fUML) applies a transformation to communicating sequential process (CSP). Then, the CSP representation is verified by a specialized tool.

3 Initial Approach

Execution and verification of models is the cornerstone of any model-driven development (MDD). One prominent alternative for MDD is model-driven architecture (MDA) established by OMG [22]. MDA defines three levels of abstraction:

(A) Computational Independent Model (CIM): to focus on the environment of the mission and mission's requirements;

(B) Platform Independent Model (PIM): to define requirements, structure, and behavior for candidate abstract solutions;

(C) PSM (Platform Specification Model): to describe concrete solutions.

MDA established a large number of specifications, but for this paper, the most important one is Alf [26]. Alf is the concrete syntax for the abstract action language defined by fUML [24], i.e., a subset of UML [23]. The execution semantics for Alf is therefore given directly by fUML. According to INCOSE [27], fUML and Alf are MDA pillars for the definition of PIMs.

fUML [24], which defines the semantics for Alf, is designed to support more than one MoC. This is pursued with leaving the semantics of some elements unconstrained. These elements define aspects of concurrency and inter-object communication which work for simulation, whereas they are not suitable for formal verification. In particular, fUML does not define semantics for: (A) timing, (B) concurrency, and (C) inter-object communication.

Our initial approach is as follows: Given the semantics defined by fUML, we specialize the explicitly unconstrained elements with the purpose of precise definitions of models using Alf. In order to do this, we discuss proposed changes in the semantics of fUML. Further, we choose to discuss the semantics in an informal way, and to present a concrete additional language construct for the specialization of Alf. This additional language construct is defined using `annotation`, which is a way to identify a modification to the behavior of an annotated statement [26]. The applied approach allows early evaluation of the proposed specialization.

3.1 Timing

The timing semantics used divides the time scale in a discrete succession of instants. Similar to languages based on the synchronous-reactive MoC, each instant corresponds to one macro-step as defined in the next subsection.

3.2 Concurrency

Concurrency can be achieved in Alf using two complementary techniques: (A) multiple active objects that, in general, imply the necessity of inter-object communication; or (B) inside a given definition by the use of the annotation `@parallel`.

Active objects are the source of all behaviors in a system modeled with UML [22], SysML, fUML, and Alf. An active object is an instance of an active class. An active class must have a `ClassifierBehavior` that defines the class behavior. Each active object is executed independently, and the only way to communicate with other active objects is through signals [23].

One alternative to provide a combination of concurrency and synchrony (where computation and communication are instantaneous) is by using the synchronous-reactive MoC. According to this MoC, a program can be defined by so-called micro and macro steps. Each macro step is divided into finitely many micro steps, which are all executed in zero time and within the same variable environment (i.e., the ordering of micro steps does not influence the semantics of a model). As a consequence, the values of the variables are uniquely defined for each macro step. Macro steps correspond to reactions of reactive systems, while micro steps correspond with atomic actions, e.g., assignments of the model that implements these reactions [29].

The demarcation of macro steps was introduced by the annotation `@pausable`; it is one of two ways to define demarcation between two macro steps. The second way is the use of the `accept` statement of Alf. This annotation is designed to be used with loop constructs (`while`,

for, do while), and the semantics is as follows: after each execution of the loop body, it waits for the next macro step. It follows that all concurrent behaviors run in lockstep: they execute the actions inside the loop in zero time, and synchronize before the next iteration.

The annotation @parallel can be used to define that all the statements in the block are executed concurrently. The block does not complete execution until all statements complete their execution; i.e., there is an implicit join of the concurrent executions of the statements [26].

3.3 Inter-Object Communication

Inter-object communication in Alf is performed sending signals (SendSignalAction) to other active objects [24]. Further, this action is not blocking, i.e., an object sends a signal and continues with its execution (it does neither wait for a response nor for an acknowledgment). A signal is a specification of what can be carried. Furthermore, a signal event represents the receipt of a signal instance in an active object [24].

Signals are based on the paradigm of message passing. Furthermore, fUML provides a point-to-point (also known as unicast) message pattern [24]. A signal is sent to a receiver (active object) using a reference to it. In contrast, multicasting is required in many safety-critical systems, e.g., fault-tolerance by active redundancy [21]. Multicasting also supports the non-intrusive observation of component interactions by an independent object. Moreover, it enables a better composition.

Multicasting was introduced by an active class called MessageDispatcher that provides the service for multicast message exchange. Instances of this class work as bus transferring instances of signals between previously registered active objects, which generate events in the target active object.

Every signal handled by MessageDispatcher has a specific identifiable sender, and zero or more receivers. The set of active objects (receivers) is defined by the existence of the reception of that signal. All signals generated in the current macro-step are instantaneously available. Moreover, signals not used during a macro step are lost. It is possible to receive signals individually or as a set. Receiving a set of signals is important for those active objects that need to process all signals sent for it in the current macro-step.

4 Case Study

We evaluated our initial approach by a case study called SmartParking that is due to [13]. The points discussed in the previous section were applied to model a part of this system. The SmartParking benchmark has been chosen for three respective reasons: (1) it is a real-world cyber-physical system, (2) it can be modeled as a discrete system [13], and (3) Geng and Cassandras [13] provide a detailed concrete solution. According to [11, 14], the case study is defined using MDA. The case study focuses on aspects related to computation and communication at the PIM abstraction level. A way to cover the control aspect is presented in [13].

4.1 Mission Context and Requirements

Mission context and mission requirements were gathered and modeled in a SysML CIM Model. The mission is summarized as follows: A user inside a vehicle shall be able to request a parking space. The request for a parking space shall be evaluated considering two constraints given by the user: (a) maximum distance from current position, and (b) maximum cost that the user wants to pay. The user shall receive a response indicating the best parking space that satisfies

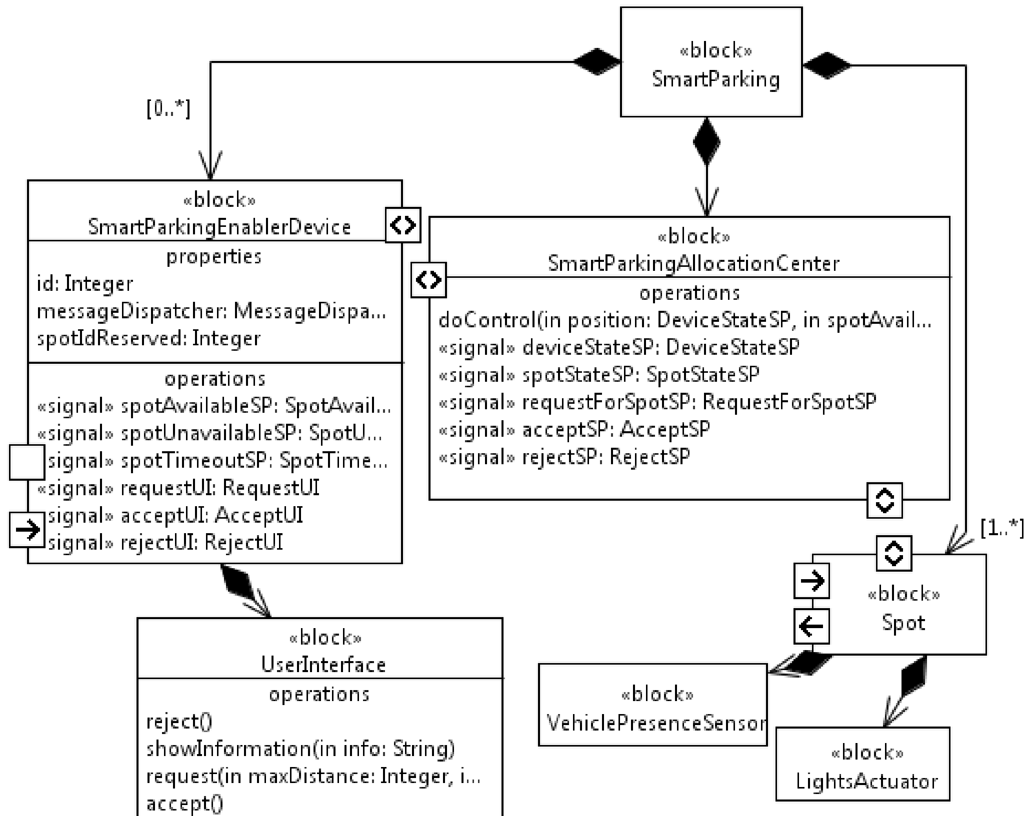

Figure 1: BDD system components (PIM level).

the imposed constraints. The user shall be able to accept or reject this response. The user shall be informed about where the parking space reserved for him/her is, as well as, about the availability of all other parking spaces up to 10 meters away from his/her current position. The vehicle shall be able to send its current position. The vehicle shall be detected when it arrives at a parking space, and when it leaves a parking space.

4.2 An Abstract Solution

Figure 1 shows the block definition diagram (BDD) for an abstract solution which is compatible with the concrete solution defined in [13]. The `SmartParking` system was decomposed in three main parts: `SmartParkingEnablerDevice`, `SmartParkingAllocationCenter`, and `Spot`. All of them are active classes.

The connections between these elements are not static. Therefore, they are not presented in Figure 1 as associations. The connections are showed in the internal block diagram (IBD) presented in Figure 2. In contrast to associations, which specify links between any instances of the associated classifiers, connectors specify links between instances playing the connected parts only [23]. The inter-object communication is provided by the multicast message exchange service (`MessageDispatcher`). Further, each active object has a reference to the same instance

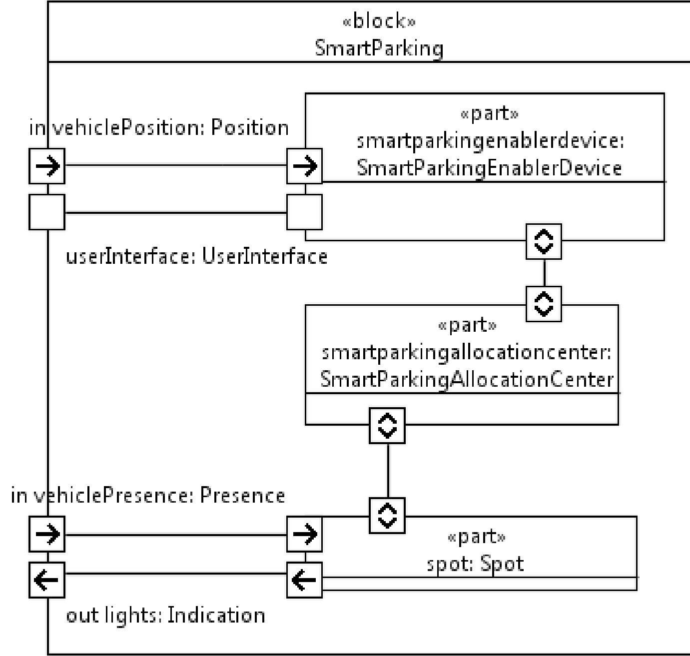

Figure 2: IBD system abstract solution (PIM level).

of `MessageDispatcher`.

`SmartParkingEnablerDevice` models a device inside the vehicle. It receives `Position` from vehicle, and has a `UserInterface` (both interactions with the environment are depicted in the upper left corner in Figure 2). Each vehicle has a corresponding `SmartParkingEnablerDevice` active object. The abstraction used in this case study makes the internal structure of this component irrelevant. It, as well as other components, could be modeled later as software, hardware or a composition of both. For example, `SmartParkingEnablerDevice` could be implemented as software in a smartphone [13].

Each parking space managed by the system is an active object `Spot`, and each `Spot` has two interactions with the environment: (a) detecting that a vehicle arrived at a `Spot` (`VehiclePresenceSensor`) and (b) indicating the current state of the `Spot` to the user, and which one is reserved for him/her (`LightsActuator`). `Spot` and `SmartParkingEnablerDevice` (plant) are both managed by a block `SmartParkingAllocationCenter` (controller). In this case study, there is only one active object of this block, which is responsible in each macro step for: (a) gathering system state and events; and (b) determining the control output.

From the viewpoint of discrete event systems (DES) control, considering signals handled by `SmartParkingAllocationCenter`, the system can be described as follows:

$$X(t) = \{D(t), P(t)\} \tag{1}$$

$$E = ECent \cup ED \cup ESpot \tag{2}$$

$$ECent = \{RequestForSpotSP, AcceptSP, RejectSP\} \tag{3}$$

$$ED = \{SpotAvailableSP, SpotUnavailableSP, SpotTimeoutSP\} \tag{4}$$

$$ESpot = \{AllocatedSP, UnallocatedSP\} \tag{5}$$

$$X(t+1) = f(X(t), U(t), W(t)) \tag{6}$$

where: Equation (1) defines the discrete state space $X(t)$ composed of $D(t) = \{k \in \mathbb{N} \mid$ `SmartParkingEnablerDevice` k in the system$\}$ (determined in each macro step by signal events of the signal `DeviceStateSP`) and $P(t) = \{k \in \mathbb{N} \mid$ `Spot` k in the system$\}$ (determined in each macro step by signal events of the signal `SpotStateSP`). Equation (2) determines the discrete event set which is composed of signals $ECent$, ED, and $ESpot$ as defined by the next equations: $ECent$ (equation (3)) is received from `SmartParkingEnablerDevice`; ED (equation (4)) is sent to `SmartParkingEnablerDevice`, and $ESpot$ (equation (5)) is sent to `Spot`. Equation (6) defines the evolution of the system over time, where the state X in the next macro step $(t+1)$ is defined by the current state $X(t)$, the current events $W(t)$, and the current control signals $U(t)$, where $W(t) = \{k \in \mathbb{N}, i \in ECent \mid \text{instance } i_k\}$ is determined by instances of signals defined in the set $ECent$, and $U(t) = \{k \in \mathbb{N}, i \in ED \vee i \in ESpot \mid \text{instance } i_k\}$ is determined by instances of the signals defined in the sets ED and $ESpot$.

Figure 3 shows that the Alf `ClassifierBehavior` of the `SmartParkingEnablerDevice` has two concurrent infinite loops. The first infinite loop depicted in Figure 3 is annotated with `@pausable`, which means that it sends the current state of device. Thereupon, it waits for the next macro step (synchronization point, before next iteration). The current state is composed by the actual position and the state of current reservation, and is represented by an instance of the signal `DeviceStateSP`. Each active object sends this signal in each macro step using an instance of `MessageDispatcher` that is responsible for delivering a copy of these messages to every registered active object that has a reception for this signal.

The second infinite loop defines the expected reactions of the device for events received from `UserInterface` and from `SmartParkingAllocationCenter`. It starts with an `accept` statement, which blocks execution (possible during many macro steps) until the expected event occurs. Subsequently, it uses the same mechanisms described above to send signals for other active objects. Moreover, it uses a compound `accept` statement that determines which block will be activated based on the type of the signal received from `UserInterface` and from `SmartParkingAllocationCenter`.

The `SmartParkingAllocationCenter` behavior is shown in Figure 4. It has an infinite loop annotated with `@pausable` that defines a synchronization point at the end of each execution of the loop body. The loop body starts with five concurrent `accept` statements, which means that it waits until no more signals of these types can be generated. Later, it applies the control law, and sends the response for other active objects (`SmartParkingEnablerDevice` and `Spot`) using the mechanism described above.

The Alf `ClassifierBehavior` of the `Spot` and the `SmartParkingEnablerDevice` are organized in the same way. There are two concurrent infinite loops: one sending signals about its state (with a synchronization point defined using `@pausable`), and another one defining reactions for the received events from `VehiclePresenceSensor` and from `SmartParkingAllocationCenter`.

```
//@parallel
{
    {
        //@pausable
        while (true){
           this.messageDispatcher.send(this,
              new DeviceStateSP(this.installedIn.positon, this.spotIdReserved));
        }
    }
    {
        while (true){
            accept(req:RequestUI);

            // sending request to SmartParkingAllocationCenter
            // through MessageDispatcher
            this.messageDispatcher.send(this,
               new RequestForSpotSP(this.installedIn.positon,
                                       req.maxCost, req.maxDistance));

            accept(spotA:SpotAvailableSP) {
                this.userInterface.showInformation(spotA.detailedDescription);

                accept(AcceptUI) {
                    // sending acceptance to SmartParkingAllocationCenter
                     this.messageDispatcher.send(this, new AcceptSmartParking());
                     this.spotIdReserved = spotA.spotId;
                     this.userInterface.showInformation("Spot accepted!");
                   } or accept(RejectUI) {
                    // sending rejection to SmartParkingAllocationCenter
                     this.messageDispatcher.send(this, new RejectSmartParking());
                     this.userInterface.showInformation("Spot rejected!");
                   } or accept(SpotTimeoutSP){
                     this.userInterface.showInformation(
                                "The maximum time for confirmation was reached!");
                   }

               } or accept(spotU:SpotUnavailableSP) {
                this.userInterface.showInformation("Spot Unavailable!");

               }

        }
    }
}
```

Figure 3: Alf `ClassifierBehavior` of `SmartParkingEnablerDevice`.

```
//@pausable
while (true) {
    //@parallel
    {
        // x(t)
        accept(deviceState:Set<DeviceStateSP>);
        accept(spotState:Set<SpotStateSP>);

        // w(t)
        // available from signals sent in current MACRO STEP
        accept(requestForSpot:Set<RequestForSpotSP>);
        accept(spotAccepted:Set<AcceptSP>);
        accept(spotRejected:Set<RejectSP>);
    }

    // u(t)
    // will be computed during doControl call
    Set<SpotAvailableSP> spotAvailableF;
    Set<SpotUnavailableSP>  spotUnavailableF;
    Set<SpotTimeoutSP> spotTimeoutF;
    Set<VehicleIsNearSP> vehicleNear;
    Set<AllocatedSP> allocated;
    Set<UnallocatedSP> unallocated;

    // applies control law
    this.doControl(deviceState, spotState,
              requestForSpot, spotAccepted, spotRejected,
              spotAvailableF, spotUnavailableF, spotTimeoutF,
              vehicleIsNear, allocated, unallocated);

    // send signals to respective active objects
    for (signal in spotAvailableF) {
        this.messageDispatcher.send(this, signal.sender, signal);
    }
    ...
}
```

Figure 4: Alf `ClassifierBehavior` of `SmartParkingAllocationCenter`.

5 Discussion

The case study defines an abstract solution (PIM) for the mission that was modeled to explore concurrency, synchronization, and multicast messages. The solution is neither complete nor optimized, e.g., signals can be removed by a centralized version of the state of the system. A tradeoff could be evaluated taking into account an objective function defined at CIM level, e.g., considering the analysis of the messages (communication) during macro steps. In addition, the abstract solution has an important difference compared to the solution presented in [13]: there are no queues. This is a consequence of the synchronous-reactive MoC: All signals are received and processed in the same macro step. The `SmartParkingEnablerDevice` does not have the state "Waiting for Assignment" [13] because, given a macro step, the system state is gathered instantaneously. Afterwards, the control law is applied and all active objects in `SmartParking`

immediately receive an adequate response.

From the viewpoint of DES control [10], the case study satisfies the following key properties: (a) its state space is a discrete set, as defined in (1) and (b) the state transition mechanism is event-driven, which means that the state can only change as a result of asynchronously occurring instantaneous events over time [10]. Apart from that, the second property has a time window to occur during a macro step. In the case study, it is mandatory that many events occur in the same macro step, and the resulting state transition reflects the occurrence of all. However, some combinations of signals in the same macro step are not allowed, e.g., if a naive device sends in a given macro step one signal for requesting a spot, and one signal for acceptance, then the last one will be lost.

Concerning modeling, state machines and state machine diagrams are commonly used for modeling state-dependent behavior. A variation of these diagrams is used to express state-dependent behavior in [13]. However, UML, fUML, SysML, and Alf do not define precise semantics for state machines [12, 30]. This is ratified by Alf, which states that a normative semantic integration of state machines with Alf will be formalized later as a part of future standards [26]. Indeed, environments of synchronous languages offer tools to visualize the resulting automata from a given textual representation [6], e.g. Figure 3 can be automatically transformed in a state machine diagram. Languages have been developed to conciliate precise semantics and automata visual modeling as e.g. [2, 18].

The nondeterminism in the fUML MoC, which was recognized by Benyahia et. al. [5], can be removed using the proposed specialization. In fact, the proposed specialization adheres the idea of introducing the synchronous-reactive MoC during early stages of a system development [4]. It avoids asynchronous complexity in early stages of system modeling, analyzing, and verification. Furthermore, the synchronous-reactive MoC enables abstract solutions to be synthesized [28] in a concrete solution using globally asynchronous locally synchronous architectures (GALS) [20], or physically asynchronous locally synchronous architectures (PALS) [19].

The initial approach presented here provides rather a starting point than a complete result. It informally defines the semantics for two complementary constructs for Alf that together can transform Alf into a synchronous action language. However, the changes needed in the fUML execution model to support it must be defined, and the points about nondeterminism stated in [5] have to be addressed.

CPS is about the intersection of the computation, communication, and control [17]. The initial approach focuses on the computational and communicational aspects of CPSs, and it can be composed with control. The case study shows that our initial approach can transfer the solid mathematical foundation of synchronous languages to SysML executable models. We consider this step, as an intermediary step, before a formal verification of executable discrete SysML models.

6 Conclusions

This paper shows the initial results of our research that has the following basic hypothesis: A specialization of Alf according to the synchronous-reactive MoC can be sufficiently expressive to model the discrete behavior of CPSs systems using SysML. These results show that the proposed specialization does not add complexity to the task of modeling using SysML, and enables concise and precise behavior definition. We believe that specializing well-known vendor-independent specifications (Alf and SysML) can provide an understandable set of languages for modeling, analyzing and verification of CPSs. Moreover, such a set of languages can enable formal verification for discrete parts of CPSs.

Acknowledgments

This work was supported by the Brazilian Coordination for Enhancement of Higher Education Personnel (CAPES) and German Academic Exchange Service (DAAD).

References

[1] I. Abdelhalim, S. Schneider, and H. Treharne. An optimization approach for effective formalized fUML model checking. In G. Eleftherakis, M. Hinchey, and M. Holcombe, editors, *Software Engineering and Formal Methods (SEFM)*, volume 7504 of *LNCS*, pages 248–262, Thessaloniki, Greece, 2012. Springer.

[2] C. André. SyncCharts: A visual representation of reactive behaviors. Research Report tr95-52, University of Nice, Sophia Antipolis, France, 1995.

[3] K. Bauer. *A New Modelling Language for Cyber-physical Systems.* PhD thesis, Department of Computer Science, University of Kaiserslautern, Germany, Kaiserslautern, Germany, January 2012. PhD.

[4] A. Benveniste, P. Caspi, S. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone. The synchronous languages twelve years later. *Proceedings of the IEEE*, 91(1):64–83, 2003.

[5] A. Benyahia, A. Cuccuru, S. Taha, F. Terrier, F. Boulanger, and S. Gérard. Extending the standard execution model of UML for real-time systems. In M. Hinchey, B. Kleinjohann, L. Kleinjohann, P.A. Lindsay, F.J. Rammig, J. Timmis, and M. Wolf, editors, *Distributed and Parallel Embedded Systems (DIPES)*, volume 329 of *IFIP Advances in Information and Communication Technology*, pages 43–54, Brisbane, Australia, 2010. Springer.

[6] G. Berry. The Esterel v5.91 system manual, 2000.

[7] E. Bousse, D. Mentré, B. Combemale, B. Baudry, and K. Takaya. Aligning SysML with the B method to provide verification and validation for systems engineering. In *Model-Driven Engineering, Verification, and Validation (MoDeVVa)*, Innsbruck, Austria, 2012.

[8] L.P. Carloni, M.D. Di Benedetto, R. Passerone, A. Pinto, and A. Sangiovanni-Vincentelli. Modeling techniques, programming languages, and design toolsets for hybrid systems, 2004. Report on the Columbus Project, http://www.columbus.gr.

[9] R. Cartwright, K. Kelly, F. Koushanfar, and W. Taha. An approach to model-driven architecture applied to hybrid systems. In *NSF Workshop on Cyber-Physical Systems*, Austin, Texas, USA, 2006.

[10] C.G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems.* Springer, 2 edition, 2008.

[11] R. Cloutier. MDA for systems engineering -– why should we care? `http://www.calimar.com/Papers/Model%20Driven%20Architecture%20for%20SE-Why%20Care.pdf`, 2006. Access date: 25.Jun.2010.

[12] H. Fecher, J. Schönborn, M. Kyas, and W.-P. de Roever. 29 new unclarities in the semantics of UML 2.0 state machines. In K.-K. Lau and R. Banach, editors, *International Conference on Formal Engineering Methods (ICFEM)*, volume 3785 of *LNCS*, pages 52–65, Manchester, England, UK, 2005. Springer.

[13] Y. Geng and C.G. Cassandras. A new "smart parking" system based on optimal resource allocation and reservations. In *Intelligent Transportation Systems (ITSC)*, pages 979–984, Washington, DC, USA, 2011. IEEE Computer Society.

[14] A. Gerlinger Romero and F.M. Gonçalves Vieira. An approach to model-driven architecture applied to hybrid systems. In *International Conference on Space Operations (SpaceOps)*, Stockholm, Sweden, 2012.

[15] H. Graves and Y. Bijan. Using formal methods with SysML in aerospace design and engineering. *Annals of Mathematics and Artificial Intelligence*, 63(1):53–102, September 2011.

[16] H. Hussmann. Loose semantics for UML/OCL. In H. Ehrig, B.J. Krämer, and A.Ertas, editors, *Integrated Design and Process Technology (IDPT)*. Society for Design and Process Science, 2002.

[17] E.A. Lee and S.A. Seshia. *Introduction to Embedded Systems – A Cyber-Physical Systems Approach*. http://leeseshia.org, 2011. ISBN 978-0-557-70857-4.

[18] F. Maraninchi. The Argos language: Graphical representation of automata and description of reactive systems. In *Visual Languages*. IEEE Computer Society, 1991.

[19] J. Meseguer and P.C. Ölveczky. Formalization and correctness of the PALS architectural pattern for distributed real-time systems. In J.S. Dong and H. Zhu, editors, *Formal Methods and Software Engineering*, volume 6447 of *LNCS*, pages 303–320, Shanghai, China, 2010. Springer.

[20] S.P. Miller, M.W. Whalen, D. O'Brien, M.P. Heimdahl, and A. Joshi. A methodology for the design and verification of globally asynchronous/locally synchronous architectures. Technical Report NASA/CR-2005-213912, National Aeronautics and Space Administration (NASA), Langley Research Center, September 2005.

[21] R. Obermaisser and H. Kopetz, editors. *GENESYS: An ARTEMIS Cross-Domain Reference Architecture for Embedded Systems*. Südwestdeutscher Verlag fuer Hochschulschriften, 2009.

[22] Object Management Group (OMG). Model-driven architecture. `http://www.omg.org/mda`, 2003. Access date: 25.Jun.2010.

[23] Object Management Group (OMG). Unified modeling language superstructure: Version: 2.4.1. `http://www.omg.org/spec/UML/2.4.1/`, 2011. Access date: 14.Apr.2013.

[24] Object Management Group (OMG). Semantics of a foundational subset for executable UML models: Version 1.1 RTF beta. `http://www.omg.org/spec/FUML/`, 2012. Access date: 24.Apr.2013.

[25] Object Management Group (OMG). Systems modeling language: Version: 1.3. `http://www.omgsysml.org/`, 2012. Access date: 27.Apr.2013.

[26] Object Management Group (OMG). Concrete syntax for UML action language (action language for foundational UML - ALF): Version: 1.0.1 - beta. `http://www.omg.org/spec/ALF/`, 2013. Access date: 27.Apr.2013.

[27] International Council on Systems Engineering (INCOSE). Survey of model-based systems engineering (MBSE) methodologies. `<http://www.incose.org/productspubs/pdf/techdata/MTTC/MBSE_Methodology_Survey_2008-0610_RevB-JAE2.pdf>`, 2008. Access date: 25 jun. 2010.

[28] D. Potop-Butucaru and B. Caillaud. Correct-by-construction asynchronous implementation of modular synchronous specifications. *Fundamenta Informaticae*, 78(1):131–159, 2007.

[29] K. Schneider. The synchronous programming language Quartz. Internal Report 375, Department of Computer Science, University of Kaiserslautern, Kaiserslautern, Germany, December 2009.

[30] M. von der Beeck. A comparison of Statecharts variants. In H. Langmaack, W.-P. de Roever, and J. Vytopil, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT)*, volume 863 of *LNCS*, pages 128–148, Lübeck, Germany, 1994. Springer.

[31] T. Weilkiens. *Systems Engineering with SysML/UML*. Elsevier, 2008.

Robotic car-like vehicles: a case study for cyberphysical systems

Federico Moro[1], Tizar Rizano[1], Daniele Fontanelli[1] and Luigi Palopoli[1]

DISI, Universitá degli Studi di Trento, Italy

1 Introduction

The technological achievements of the Information and Communication Technology are reshaping our life. The constant access to the network offered by the new generation of mobile devices discloses important and unprecedented opportunities for business users and for "simple" consumers alike. The next thrust is expected to come from the so-called cyberphysical systems (CPS) [4].

A CPS is in the common lingo a device or a system where the computation units are deeply interconnected with the physical system they control. This definition is apparently very close to that of a "classic" embedded system (ES). However, CPSs are usually characterized by a heterogeneous and distributed architecture and, more frequently, have the ability to share information and services with other CPSs disseminated in the environment setting the basis for an "internet of things" [1]. Another difference is the number and complexity of control functions and their interconnection which is supported by a variety of sophisticated sensing devices and the related perception algorithms. The overall complexity is due to the need for a high degree of autonomy, and for reconfiguration and adaptation capabilities which provide robustness to changing and unanticipated environment conditions.

We have just outlined some of the requirements posed to the upcoming generation of CPSs, but they are sufficient to suggest the challenging difficulty of the development. We believe that this level of complexity requires out-and-out science of CPSs, where some of the traditional methodologies developed for ESs will be revisited and integrated with new ideas and paradigms [3]. What we find it is lacking in the literature is the presence of system examples to use as benchmark for those methodologies. Available examples tend to be artificial and do not cover "pratical" aspects of the application. There is a need for case studies which are easy to replicate in research labs.

The objective of this paper is to offer one of these examples which is a trade-off between complexity and tractability. The case study could support, as benchmark, different research activities, such as: 1) automated-mapping of functional models onto software/hardware architecture (e.g., as shown by Zheng et al. [8], 2) end-to-end design of distributed real-time systems, 3) specification languages and planning for autonomous robots, 4) resource aware control (e.g, [7]).

In Section II we describe the system giving details about the hardware and software used in the architecture, the set of sensors and actuators and the consequent description of the capabilities of the system, and an overall view of the functional scheme. In particular, Section II gives some hints for potential research topics which could find applicability in our case study. In Section III we conclude the paper outlining the structure and content of the website that will be used for distribution of the case study.

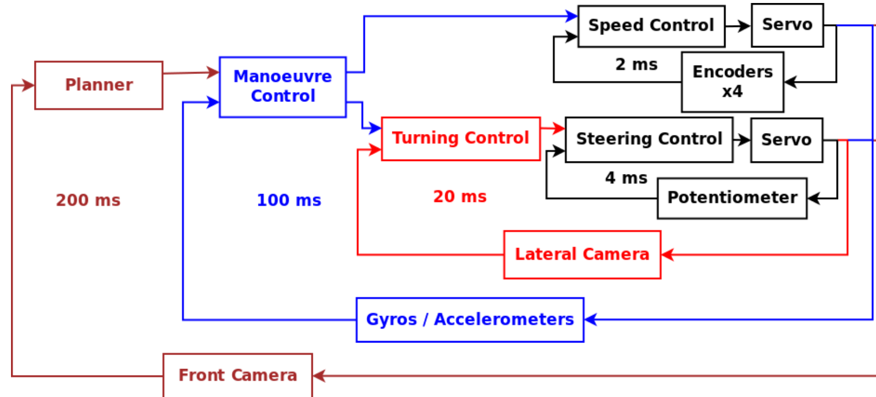

Figure 1: Functional organization of the system. Sensing and actuating blocks are involved in control loops. Contollers are connected in a nested structure, with the planner being the driving block of the entire system. Each loop is associated with a period.

2 System Description

The case study considered in this paper is deeply inspired by the automotive industry. We consider a robotic car, which we have prototyped using a scaled model in our laboratory. In the following text, we will first describe the main functional components of the car. Then we will move on to describing the hardware/software architecture used for its implementation.

Functional View. The system can be considered as a particular instance of a much larger class of similar cyberphysical systems. It is interconnected to a physical environment, which is comprised of the road where the car moves (along with external objects and agents) and by the physical components of the system. Sensors collect information (in our case position, attitude and velocity of the car), while actuators are the means that the system has for environment manipulation (in our case the engine and the steering wheels).

Sensors and actuators are involved in the computational activities that implement the feedback control loops for system stabilisation, which in our case enable the system to follow a predefined line. The complexity of the system requires a *planner*, which decides the system goals and supervises their fulfilment. In the presented case, the planner decides how to follow the line, in essence a sequence of manoeuvres such as go straight with speed X, turn with radius Y and speed Z etc. This decision is determined by different considerations, such as saving time, saving energy, overtaking slower vehicles, avoiding fixed obstacles etc. In order to have a full control over the system, a cooperation between planning and sensing part has to be established. The planned path is tracked by the low level controllers.

From a high level perspective, the system comprises a set of nested control loops. Each control loop is activated periodically and has a different frequency, as shown in Fig. 1. Analyzing the model from a closer view point, at the low level the robot is equipped with two servos: one is the engine used to move the car (which operates on the four wheels), and one is used to control the steering angle. The two servos are controlled by a PWM signal. Each wheel has a relative encoder for speed monitoring, and, on the front of the car, a potentiometer is mounted in order to get a feedback of the steering position. Such sensors are used by two low level feedback loops (activated with a period of 2 ms and 4 ms), which implement PID controllers used to regulate the speed and the turning angle. A basic Inertial Platform, composed by gyros

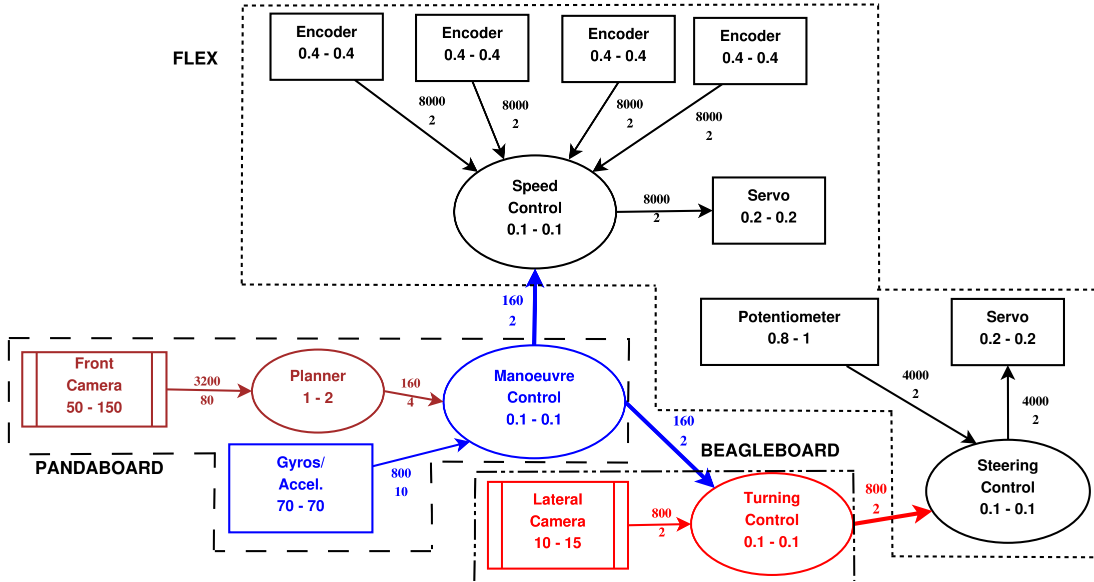

Figure 2: The task set generated from the functional diagram. Each node represents a task and is labeled with its average and worst case execution, both in ms. Edges represent communication between tasks, and bitrate (bit/s) and payload (bytes) are specified. The diagram also shows the allocation of the tasks in the computing units used in the system.

and accelerometers, completes the set of low level sensors and is used to improve the estimate of the car position.

The information collected in the low level are fed to a line following algorithm that controls the position of the car with respect to its ideal trajectory. The algorithm utilises a high frame rate camera, pointing sideways and used to estimate position and attitude of the car with respect to the road line. The line following algorithm and the speed controller receive set points from a manoeuvre controller that decides the sequence of manoeuvres and monitors their execution using encoders and the inertial platform to estimate the progress along the planned line.

A second camera, mounted on the front of the vehicle, is used for path reconstruction and obstacles detection. This camera is activated with a relatively low rate (5 frame per seconds). The Planner receives an image captured through the camera, reconstructs the path and extracts other meaningful information (e.g., obstacles). The Planner, therefore, decides which manoeuvre the vehicle has to perform and communicates it to the Manoeuvre Controller.

The vision algorithms used for each camera use a combination of Randomised algorithms (RANSAC) and Kalman Filtering [5, 6]. Such sensing activities generate a widely changing computing workload, a situation difficult to manage with the standard tools of digital control [2].

Hardware Architecture. The computing system is a balanced blend of microcontrollers and microprocessors. There are three main components. The first one is the FLEX: a development board based on a 16 bit dsPIC. The board is provided with a complete software infrastructure based on Erika which is a RTOS OSEK compliant. The PIC technology allows developers to interface the microcontroller with external objects by means of common digital interfaces like low power communication systems (SPI or I^2C), or PWM. The FLEX also supports advanced communication technologies like Ethernet and CAN bus.

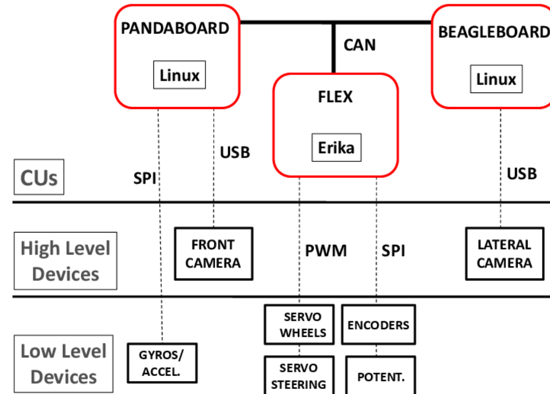

Figure 3: The hardware architecture of the system. Pandaboard, Beagleboard and FLEX are connected with a CAN bus, while each board has specific connections for communication with sensors and actuators.

The other two components are two ARM–based evaluation boards: Beagleboard and Pandaboard. Both are Texas Instruments products and are based respectively on OMAP3 and OMAP4 processors version. The RTOS used for these components is a Linux kernel modified with RTpreempt patches, which improve its real–time performance. The two boards export SPI and I^2C interfaces for connection with sensors and other low level peripheral; other connectivity solutions are the classic USB, Ethernet, Bluetooth and IEEE802.11 which facilitate remote control and telemetry. Such boards have a sufficient computing power to support the functional blocks described above, but have a limited power consumption and a low cost, both desirable features for autonomous mobile robots.

Fig. 3 shows the overall picture of the architecture. The processing units are connected through a CAN BUS, which offers a sufficient bit-rate for the applications at hand while avoids the power consumption of an Ethernet switch. Other communication technologies are used for sensors and actuators interface. In the proposed setting, the FLEX board is adopted for the low level functionalities of the system: motor and steering controllers, and communication with sensors. Nevertheless, gyros and accelerometers requires the connection with the Pandaboard due to the high communication rate required, which would be unsustainable by the FLEX.

The two cameras, considered as high level devices, have an USB connection to the two ARM boards. Usually, ARM processors design integrates in the system some co-processing units, like GPUs or DSPs, which extend the computational capabilities of this kind of platforms. Using this extra availability of computing power it is then possible to increase the performance of executed algorithms, in particular of the randomized algorithms used in this study case. This reinforces the choice of using microcontrollers to define a level of sensors and actuators interaction, and exploit more advanced processors for the high level intelligence of the system. A common bus between the processing units facilitates the data flow for all the tasks running in the system.

Software Architecture and Mapping. Model based approaches recommend to use such models as the primary design primitives to fine tune the design of the planner and of the control algorithms. The models composing the functional diagram, which are essential for unveiling the mathematical and physical aspects related to the stability of the system and to the correctness of the design, suggest a possible decomposition into subsystems, a definition of their relations and of the timing constraints for their execution.

After the system functionalities and time constraints are defined, the computational entities are generated through automated tools or by a manual coding process. This is a refinement step that produces a set of concurrent tasks, each one with a period related to the control loop for which the task is involved. The set of tasks is easy to represent with a directed acyclic graph (DAG), where the nodes correspond to the tasks and the edges are the involved communications. An edge is associated with a weight which specifies the amount of data that need to be sent (bitrate requirements). Fig. 2 shows how a DAG could be derived from the previous diagram in Fig. 1. In our simple hypotheses, every functional block generates a task which receives and sends messages to the tasks derived from the other functional blocks involved in the same control loop. If required, some tasks may be further refined into subtasks. For instance, the Lateral Camera task could be split in two tasks: one for frame capturing and one for image processing. The splitting increases the complexity of the scheduling problem, but could give benefits from the overall computational power utilization. Furthermore, the allocation of the two tasks in different computing units adds a new message in the system, that is in this example a whole frame.

Once a refinement of the system into tasks has been produced, the next steps is to map them into our hardware architecture in order to take advantage of the physical parallelism enabled by the different computation units and by their interconnection buses. An optimized task allocation guarantees that all the tasks respect their deadlines and that the traffic generated by tasks communication is sustainable by the system. For this reason, the allocation of tasks to a specific computing unit not only should consider the computational power of the CPU, but also the bandwidth required by output emission of the tasks. Tasks that need to communicate, if allocated in the same unit, will not increase the traffic in the system because the data exchange will happen internally at the computational unit. Whatever is the communication link between the two tasks, a fundamental problem in this case is to ensure that the system obtained has the same semantics of its abstract counter part.

3 Research Topics

The image processing algorithms, in particular the one that reconstructs the path, have a time of execution with a high variability. This makes the hard real-time approaches to control design difficult and discouraging. The case study is well suited for methodologies for soft real-time control design, that is the design of control schemes that are tolerant to varying delays. In this kind of systems usually is required to find a trade-off between control performance and resource utilization.

As described in the previous section, the functional scheme is eventually transformed in a set of tasks which need to be allocated and scheduled in the computing units. Modeling of task sets and mapping algorithms could be tested in our benchmark. The mapping result has to meet the time requirements and be computationally feasible in the hardware architecture.

The planning subsystem is also an interest subject for research. In particular, our application needs the identification of an alphabet of manouvres, and relative synthesis algorithm to generate winning and safe plans, accounting for computation time and vehicle performance. More in general, the system could be used for testing planning algorithms that express missions where dynamic events, prioritized goals, and probability of failures are considered.

4 Conclusion

We have described our case study and listed some interesting research activities which could use our system as benchmark. The next step is to create a web space in which we provide information about where to buy the car and the hardware used to create the final system, together with cabling details and design schemes of PCBs. We will provide software code with open-source license, and configuration details of drivers and operating system. The idea is to create a guide that helps other researchers to replicate the system in a reasonable amount of time.

Finally, we will provide simulators and potential solutions to the different aspects of the system, while collecting solutions of other researchers that are willing to share and compare their results.

References

[1] L. Atzori, A. Iera, and G. Morabito. The internet of things: A survey. *Computer Networks*, 54(15):2787–2805, 2010.

[2] D. Fontanelli, L. Greco, and L. Palopoli. Soft RealTime Scheduling for Embedded Control Systems. *Automatica*, 2013. To Appear.

[3] E. A. Lee. Cyber-physical systems-are computing foundations adequate. In *Position Paper for NSF Workshop On Cyber-Physical Systems: Research Motivation, Techniques and Roadmap*, volume 2. Citeseer, 2006.

[4] E. A. Lee. Cyber physical systems: Design challenges. In *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*, pages 363–369. IEEE, 2008.

[5] F. Moro, D. Fontanelli, and L. Palopoli. Vision-based robust localization for vehicles. In *Instrumentation and Measurement Technology Conference (I2MTC), 2012 IEEE International*, pages 553–558. IEEE, 2012.

[6] D. Nistér. Preemptive ransac for live structure and motion estimation. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 199–206. IEEE, 2003.

[7] A. Quagli, D. Fontanelli, L. Greco, L. Palopoli, and A. Bicchi. Design of Embedded Controllers Based on Anytime Computing. *IEEE Trans. on Industrial Informatics*, 6(4):492–502, November 2010.

[8] W. Zheng, M. Di Natale, C. Pinello, P. Giusto, and A. S. Vincentelli. Synthesis of task and message activation models in real-time distributed automotive systems. In *Proceedings of the conference on Design, automation and test in Europe*, pages 93–98. EDA Consortium, 2007.

Towards Dynamic Deployment Calculation for Extensible Systems using SMT-Solvers

Klaus Becker[1] and Sebastian Voss[1]

Software and Systems Engineering
fortiss GmbH
Guerickestr. 25, 80805 Munich, Germany
becker@fortiss.org, voss@fortiss.org

Abstract

During design of distributed embedded systems, the determination of the deployment of software components to execution units is a crucial subtask of the design space exploration. In static systems, the deployment can be determined at design time. However, in many cases it is desired to add new functional features into existing systems after sale. In this case, new software components have to be integrated into the former system and hence, into an existing deployment.

We aim in a self-configuring system that ensures the integrity of the integration of new components autonomously. Our proposed process of integrating new components into a given system consists of several steps intended to be applied at run-time while the system is in operation. Beside others, the process includes a logical admission control, followed by a self-configuration of the system.

In this paper, we focus on extending an existent deployment during the self-configuration phase incrementally. We sketch a mechanism that extends existing deployments with additional components in an efficient way by using SMT-solvers. We also present an example that demonstrates how these solutions are calculated based on given deployment-constraints.

Contents

1 Introduction and Motivation **2**

2 The process of integrating new functionality **2**

3 Assumed properties of the underlying system **4**

4 Application design **4**

5 On-line design space exploration for self-configuration **5**

5.1 Constraints for the Design Space 5
5.2 Self-Configuration Process 5
5.3 Deployment Problem 5
5.4 Solution Model 6
5.5 Example 6
5.6 Extending the Example 8

6 Related work **8**

7 Conclusion and future work **9**

8 Acknowledgments **10**

1 Introduction and Motivation

Maintainability and extensibility are important properties of long living systems, especially also in distributed embedded systems. After a system is taken into operation, there might arise several reasons to maintain the system in a functional or non-functional manner. Non-functional maintenance tasks change the system without changing the systems functional features experienced by the user. Functional maintenance affects the user-experience, e.g. by updating or extending the system functionality. Updates or extensions might be required in case of changing or new requirements, or due to changes in the environment. In this paper, we focus on extending functionality of component based systems without requiring a temporal shutdown of the system under maintenance.

In this paper, we tackle the problem of extending distributed component-based embedded systems with new components that realize new functional features. We focus on the design space exploration for the new components, more precisely on the determination of the deployment. During deployment determination it is defined which software component is executed on which execution unit, while considering different constraints that must be hold by the deployment in order to be valid. This is just one sub-step of the integration process of new components. We developed a deployment calculator which is based on the usage of a SMT-Solver.

We aim in executing these techniques at runtime by the system under maintenance itself, in order to let the system have control about its own integrity. This is for instance useful for distributedly developed systems, in which no central authority performs and proofs the integration of new functional features, but also for systems that cannot be taken out of operation completely in order to install new functionalities. Examples of such systems are production lines where each production stop denotes huge costs, remotely maintained systems which cannot be accessed by humans, but also future automobiles in which new functional features are desired to be installed after sale without having to go to a workshop.

We show a methodology to find deployments in a incremental manner. Incremental means that an existing deployment is extended with new components. The requirements are that the former system components should not be affected by the new components negatively, as well as that the delta between the configuration of the old and the new extended system should be as small as possible. In case of deployment, this means that the deployment of existing components should not change when new components are added. This is to avoid on-line migrations of components. Between the extension phases, the system operates in a static manner.

The remainder of this paper is as follows. In section 2, we show the big picture on our work, which is the intended process for integrating new functions into existing systems. As our integration process is supported by properties of the underlying platform on which it is performed, we discuss these properties briefly in section 3. In section 4, we show briefly how the components are designed that are intended to be added to the system. Section 5 shows then a sketch for an incremental deployment calculation during the self-configuration phase. This is also shown by an example with some example deployment constraints. Finally, section 6 discusses related work and 7 the conclusion and future work.

2 The process of integrating new functionality

We aim in an integration process including an on-line admission control for the new functionality followed by a self-configuration. During integration, it has to be ensured that the system keeps operating conform to its specification. This is especially important for safety-critical real-time systems, because a loss of integrity might cause hazardous damage to material and life. To

provide new system functionality in a plug-and-play manner, the system has to be able to verify autonomously if the new functionality can be integrated or not. However, as mentioned above, in this paper we focus on the deployment calculation which is applied during the self-configuration phase of the integration process.

Fig. 1 shows our intended five main steps that are required to integrate new software components into a given system.

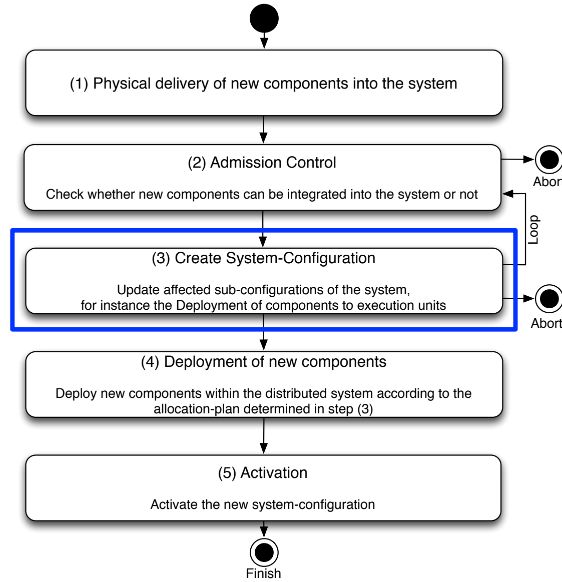

Figure 1: Activities during the extension of the system

In phase (1), the new components are physically made available to the system. Phase (2) performs a logical admission control. Here, different formal analysis methods can be applied, which are normally applied at design time of static systems, e.g. to check interaction between components. Also compatibility and dependencies on feature level can be checked here, as well as guaranteeing security by checking certificates. However, this phase is out of scope of this paper.

If phase (2) successes, the new components can be added from logical point of view. Hence, a new system configuration has to be determined in phase (3) including the new components. During this, the new components are for instance integrated into the execution and communication schedules as well as into the deployment-configuration of the system. The latter point is in scope of this paper. We aim in an incremental self-configuration approach in which the old system configuration parts keep as most unchanged as possible (cf. section 5).

However, it might happen that the self-configuration is not successful. In such situations, new components can only be integrated when a subset of the existing components is either removed, degraded or migrated to another execution node in order to free enough resources to enable a valid configuration together with the new components. This is done by going back to the logical analyses phase to select a replacement strategy for existing components. However, such replacement strategies are out of scope of this paper.

After the new deployment-configuration has been determined, the new components are physically deployed in phase (4) to the target execution units. Finally, phase (5) activates the

new configuration, meaning to switch to the new schedules and hereby to activate the new components. The question of how to activate the new configuration safely without negatively affecting system service is not discussed in this paper.

All this should be possible for components that were unknown at the design time of the former system. The new components should not need to know something in advance about the system in which they are desired to be integrated. This allows highest flexibility in component composition.

3 Assumed properties of the underlying system

Our presented approach for extending systems at runtime requires support by the underlying platform. We assume some fundamental architectural drivers that are common principles.

We assume a distributed embedded system with a middleware driven distributedly accessible *data-pool* that allows indirect access to sensors and actuators. The middleware also ensures portability of the components. The portability together with the data-pool allows freedom on the allocation of the software components to the execution units, which is essential for the deployment determination.

The middleware is responsible for the transmission of sensor data into the data-pool, providing required data to the software components and delivering output data to their destinations, like physical actuators. This means, sensors and actuators are decoupled from control functions via the data-pool.

Data-dependencies are not modeled by explicit channels between components, but by specifying the required respectively provided data. The middleware has a mechanism that determines possible matches of data producers and data consumers and creates channels between the components during configuration phase, based on the data-specifications. This follows the laws of *blind communication* [7]. Due to this, components are fully exchangeable by other components that produce and require data with conforming specification. We assume also a flexible network that allows to add new network packages at runtime.

However, the concrete realization of the mentioned middleware is not further discussed in this paper as it is out of scope.

4 Application design

The on-line integration process requires some additional pieces of information about functional and non-functional properties of the components, required for admission control decisions and self-configuration. In classical static systems, these information are only required at design time. However, in a dynamically extensible system, these information are also required at runtime during the integration process. Hence, components need to be enriched with a set of information about their functional and non-functional properties. This can for example be addressed using rich components [1].

Each component contributes to realize one or more functional features of the system. We consider components as black-boxes that might however have nested invisible sub-components. Black-box components specify their external communication by defining the required or provided data at their ports. The wiring of the communication channels between black-box components is done during the integration process according to the given data specifications at the ports.

5 On-line design space exploration for self-configuration

We consider an *incremental* self-configuration approach to integrate new components into a given configuration. This comprises that the former system configuration should keep as most unchanged as possible, meaning that existing components keep their locations and new components are deployed into free gaps. It is not desired to migrate existing components between execution units at run-time. The proposed incremental approach targets for reaching the desired level of extensibility by using on-line design space exploration mechanisms.

5.1 Constraints for the Design Space

System design affects temporal and spacial issues. Therefore, we distinct *partitioning*, determining the borders of a black-box component during design time, and *allocation* (aka mapping or deployment), which means deciding the assignment of software components to hardware execution nodes and pertains to spacial requirements. Based on this, a temporal configuration is the execution order of these software components (or *tasks*) on their allocated execution nodes (aka schedule) as well as the temporal order of communications (or *messages*) on a shared communication medium.

In order to solve this kind of problem, different sets of constraints need to be considered. First of all, constraints with respect to a suitable *deployment* are considered. The allocation has to comply to the existing resource constraints of the system. For instance, software components might have allocation constraints w.r.t. a dedicated location. One further constraint might be that it is desired to partition communicative component-clusters onto the same execution nodes, in order to reduce the network load. All these constraints might conclude in a *multi-objective* optimization problem with contradicting objectives.

5.2 Self-Configuration Process

We work towards a *hierarchically* coordinated self-configuration process, enabled by the datapool (cf. Sec. 3) that provides all required pieces of information (cf. Sec. 4) and can be used to deploy the new configuration towards the distributed system nodes. Hierarchical means that there exists a master control instance for the integration process, which cooperates with the distributed system nodes in order to determine a valid holistic system configuration.

As this approach should work on-line, we propose to have *scalable* techniques for calculating new configurations. An approach may rely on a symbolic encoding scheme for the problem under consideration. Therefore, we describe it as a satisfiability problem using boolean formulas and linear arithmetic constraints. A state-of-the-art *SAT modulo theory (SMT)* solver is used to compute new configurations for such systems in a scalable manner. Satisfiability Modulo Theory (SMT) enables checking the satisfiability of logical formulas over one or more theories. The solver proves a model as a single solution. However, optimized solutions may be of a particular interest. Finding optimized solutions takes more time and requires potentially some meta-search techniques (e.g. binary search, generic algorithms) on top of the SMT-based problem [9].

5.3 Deployment Problem

As described in section 5.1, the calculation of a valid deployment comprises the assignment of a set of software components S to a set of execution nodes E, while fulfilling all given constraints. Furthermore, we target the reduction of required network traffic introduced by communication channels C between software components.

Our system model $\mathcal{M} = \langle S, C, E, \alpha \rangle$ contains a set of software components (SWCs) $S = \{s_1, s_2, ..., s_m\}$, a set of directed communication channels $C = S \times S$ between software components, a set of execution nodes $E = \{e_1, e_2, ..., e_k\}$, and an allocation $\alpha : S \to E$ that returns the set of execution nodes $e \in E$ to which a software component $s \in S$ is deployed. This can also be written as an allocation matrix $\alpha(s_i, e_j)$ returning 1 if $e_j \in \alpha(s_i)$, otherwise 0.

Furthermore, we define the following parameters for the system model artifacts:
$wcet : S \to \mathbb{N}$ defines the worst-case execution time (WCET) of software components $s \in S$,
$weight : C \to \mathbb{N}$ defines weights for communication channels $c \in C$, and
$tbudget : E \to \mathbb{N}$ corresponds to the time-budget of the $e \in E$.

A channel-weight $weight(c)$ is the communication load in bits/s introduced by the channel $c \in C$. The $tbudget(e)$ indicates how much time in ms is available to execute software components (in a given time period) on the given execution node. All these parameters are set by constraints to fixed constants regarding to a certain system model.

We assume the following further deployment constraints:

1. The sum of execution times $wcet(s)$ of SWCs deployed to the same execution node is not allowed to exceed the provided time budget $tbudget(e)$ of that execution node.

   $\forall e_j \in E \left(\sum_{s_i \in S} (\alpha(s_i, e_j) \cdot wcet(s_i)) \leq tbudget(e_j) \right)$

2. The sum of channel weights $weight(c)$ between SWCs allocated to different execution nodes E must not exceed a specified network threshold N^{Th}, defining the upper limit for the weight of network communication.

$$\sum_{c_i(s_k, s_l) \in C \mid \alpha(s_k) \neq \alpha(s_l)} weight(s_k, s_l) \leq N^{Th}$$

These constraints can be encoded into SMT formulas. The objective is to find a valid allocation α of all SWCs to the execution nodes, fulfilling all given constraints.

We encoded the parameters and constrains for the Z3 theorem prover [2]. However, the approach is not dependent on this specific SMT solver, also other solvers can be used.

5.4 Solution Model

The purpose of a SMT solver is to check the satisfiability of logical formulas over one or more theories. In our case, the provided solution model is a valid allocation α for the given deployment problem. Thus, the SMT solver returns one solution that fulfills the defined constraints. This is in general not an optimized solution regarding to some objective function, but just a valid solution for the specified constraints. The solution model consists of interpretations for the variables, functions and predicate symbols that makes the formula true. In our case, this gives a valid allocation matrix.

5.5 Example

Let the software components S and execution nodes E in the example have the following properties:
$S = \{s0, s1, s2, s3\}$
$E = \{e0, e1\}$

$wcet(S) = \{4, 4, 4, 4\}$
$tbudget(E) = \{10, 10\}$
$weight(s0, s1) = 1$
$weight(s0, s2) = 2$
$weight(s1, s2) = 4$

This is encoded as input for the SMT solver, together with a threshold N^{Th} for the maximum allowed network traffic.

For $N^{Th} = 5$, a valid solution for deployment α is shown in Fig. 2. Fig. 3 shows a solution for $N^{Th} = 4$, which can be hold only with a different deployment. The deployment for $N^{Th} = 3$ is the same as in Fig. 3. A deployment for $N^{Th} = 2$ is not feasible. To minimize the network traffic, we solved the problem multiple times with decreasing N^{Th}.

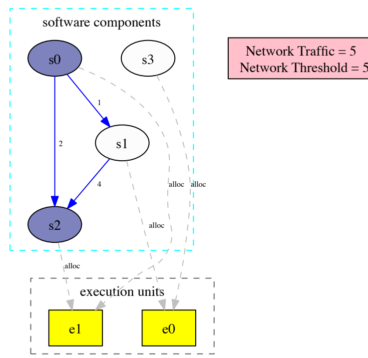

Figure 2: Solution for the deployment of 4 SWCs with 3 channels to 2 execution units, Network Traffic Threshold = 5

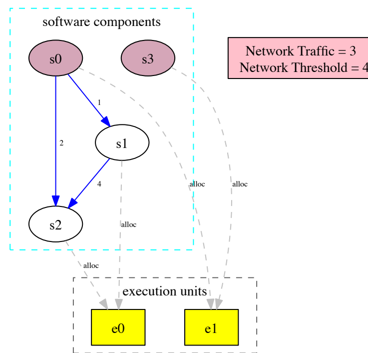

Figure 3: Solution for the deployment of 4 SWCs with 3 channels to 2 execution units, Network Traffic Threshold = 4

5.6 Extending the Example

After the deployment of the initial system has been determined, the deployment should now be extended by an additional software component $s4$, having a WCET of $2ms$ and required and provided data-specifications that force the creation of two additional channels:
$weight(s0, s4) = 1$
$weight(s4, s3) = 1$

Fig. 4 shows the deployment after the new component $s4$ has been integrated. Notice that the deployment of the existing components keep untouched. This is reached by setting the former deployment of the existing components as fixed solution constraints during the deployment calculation of the extended system.

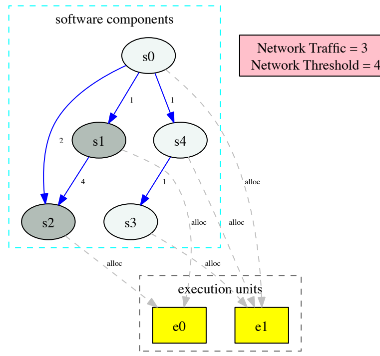

Figure 4: Solution in case an additional component was plugged in causing two new channels, Network Traffic Threshold = 4

However, it might happen that the network threshold cannot be hold for the extended system. In this case, N^{Th} has to be relaxed until a valid solution is found.

All the figures were generated by our deployment calculator by using the graphviz framework (www.graphviz.org).

6 Related work

The problem of finding optimized allocations of functions onto execution platforms (e.g. electronic control units) has for instance been considered in the following works.

In [3], an approach for centralized self-management with focus on self-configuration and self-healing in heterogeneous systems is proposed for the automotive domain. The approach uses publish/subscribe and request/response communication. As use cases, updating, installing and removing of applications are mentioned, as well as attaching and detaching platforms. Tackled self-configuration problems are the deployment of applications (resp. their components) to heterogeneous platforms. The Self-Configuration is performed by using constraint satisfaction problems (CSP). The Web ontology language (OWL) is used to describe platforms and components with information, required by the self-configuration. Two self-configuration algorithms are presented and compared by simulation, namely *backtracking* (worst-fit) and *Iterative repair* (min-conflict). The former algorithm is slower but better usable for building configurations

from scratch, while the latter algorithm is faster, independent of the number of components and better usable if a configuration for the previous system state is given.

In the project DySCAS, an automotive embedded middleware supporting extensibility was investigated. Mentioned use-cases were attaching new devices like sensors/actuators, integrating new software functionality and the shutdown of non demanded devices for power saving reasons. In case of an addition of a new task to the system, also re-allocations of existing tasks may be performed. During the deployment calculation, the aim is to maximize the total quality-of-service benefit of the tasks relative to their resource usage [8]. However, the deployment problem was solved by an algorithm and not by a more generic SMT-solver supporting a broad set of constraints in an easily usable way.

In [10], a comparison is shown about deployment-calculation by a SAT-Solver and by the Simulated Annealing Algorithm. The result was that SAT solving scales better and is more efficient for larger sets of equations. The use-case of the shown work is to find a new valid software allocation in case of a component failure. This allocation determination has to be performed as fast as possible to heal the system quickly. However, for our work we do not see the task of creating a new configuration as time-critical itself, because we apply the self-configuration only during the integration of new functionality, what we do not consider as time-critical because the system operates stable during the determination of the new configuration.

Optimisation of the allocation of functions in vehicle networks was also investigated in [5]. Self-adaptive ant colony optimisation applied to function allocation in vehicle networks was shown in [4].

Beside the deployment calculation problem, also the determination of feasible schedules is a subtask of design space exploration. An approach for the determination of static schedules of a time-triggered network-on-chip was described in [6]. The approach performs an optimization based on an evolutionary algorithm set on top of the Z3 SMT Solver.

7 Conclusion and future work

In this paper, we have shown a methodology towards supporting extensions to existing deployments in the use case to extend distributed systems with new components. Our approach is based on the usage of SMT-solvers and is intended to be applied at system-runtime in a self-configuring manner. We discussed the assumed underlying platform properties supporting our approach and presented a sketch example.

In order to cover all parts required to obtain a complete self-configuring integration process for new components, there are still a bunch of open issues to do. Important questions are for instance how to cut the configuration problems into sub-problems that can be solved independently or hierarchically and how to setup the architecture of the self-configuration itself to reach a scalable configuration? One question is also how to use freedom on open design decisions to obtain optimal configurations, like the choice of concrete channels between components during the integration process. Also appropriate replacement strategies are of interest for the case that the self-configuration was not successful, but the new components should be integrated nevertheless.

As future work, we are going to evaluate the efficiency and scalablility of our SMT-based self-configuration approach to different sets of software components and execution units.

Furthermore, we are going to refine the deployment problem for a new platform architecture for future electric vehicles that supports mixed-critical and fail-operational features. This platform fulfills our assumed properties and is going to support extensions in a self-configuring plug-and-play manner.

8 Acknowledgments

This work has been investigated in the context of the Project RACE (Robust and Reliant Automotive Computing Environment for Future eCars), funded by the German Federal Ministry of Economics and Technology (BMWi) under grant no. 01ME12009.

References

[1] W. Damm, A. Votintseva, A. Metzner, B. Josko, T. Peikenkamp, and E. Böde. Boosting re-use of embedded automotive applications through rich components. *Proc. of Foundations of Interface Technologies*, 2005.

[2] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, 2008.

[3] M. Dinkel and U. Baumgarten. Self-configuration of vehicle systems-algorithms and simulation. In *WIT'07: Proceedings of the 4th International Workshop on Intelligent Transportation*, pages 85–91, 2007.

[4] M. Förster, B. Bickel, B. Hardung, and G. Kókai. Self-adaptive ant colony optimisation applied to function allocation in vehicle networks. In *Conference on Genetic and evolutionary computation (GECCO)*, pages 1991–1998. ACM, 2007.

[5] Bernd Hardung. *Optimisation of the allocation of functions in vehicle networks*. PhD thesis, University of Erlangen-Nuremberg, 2006.

[6] J. Huang, J.O. Blech, A. Raabe, C. Buckl, and A. Knoll. Static scheduling of a time-triggered network-on-chip based on smt solving. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012*, pages 509–514. IEEE, 2012.

[7] P. Oreizy, M.M. Gorlick, R.N. Taylor, D. Heimhigner, G. Johnson, N. Medvidovic, A. Quilici, D.S. Rosenblum, and A.L. Wolf. An architecture-based approach to self-adaptive software. *Intelligent Systems and Their Applications, IEEE*, 14(3):54–62, 1999.

[8] T.N. Qureshi, M. Persson, D.J. Chen, M. Törngren, and L. Feng. Model-based development of middleware for self-configurable embedded real-time systems: Experiences from the dyscas project. 2009.

[9] S. Voss and B. Schaetz. Deployment and scheduling synthesis for mixed-critical shared-memory applications. In *Engineering of Computer-Based Systems (ECBS)*, 2013.

[10] M. Zeller, C. Prehofer, G. Weiss, D. Eilers, and R. Knorr. Towards self-adaptation in real-time, networked systems: Efficient solving of system constraints for automotive embedded systems. In *Self-Adaptive and Self-Organizing Systems (SASO)*, pages 79–88. IEEE, 2011.

On the Technological and Methodological Concepts of Federated Embedded Systems

Avenir Kobetski and Jakob Axelsson

Swedish Institute of Computer Science (SICS)
Kista, Sweden
avenir.kobetski@sics.se, jakob.axelsson@sics.se

Abstract

Traditionally embedded systems are developed with a specific control task in mind, and are able to affect only a limited set of actuators, based on measurements from a limited set of sensors. With the arrival of cheap and efficient communication technology, this traditional picture is starting to change. It is our belief that future embedded systems will interact with each other, forming federations to provide new emergent services to their users. With this in mind, a pre-study was performed to discern the main concepts of such federations and the related challenges that need to be addressed. This has led to two parallel research directions, presented in this paper. One is focusing on the enabling technology that is needed for dynamic creation of new types of federations, while the other deals with the methodological concepts for creation of ecosystems in which federations of embedded systems can be dynamically formed.

Contents

1 Introduction **1**

2 High level concepts **2**
2.1 System-level Technology Concepts 3
2.2 Federation-level Technology Concepts 3
2.3 System-level Methodology Concepts 3
2.4 Federation-level Methodology Concepts 4

3 Dynamic Software Reconfiguration in Embedded Systems **4**
3.1 The AUTOSAR Concept 4
3.2 Dynamic Software Components 5
3.3 Internal communication 6
3.4 Safety & Security 6

4 Ecosystems of embedded systems **6**

5 Related work **7**

6 Conclusions and future work **8**
6.1 Acknowledgements 9

1 Introduction

The invention of Internet revolutionized knowledge sharing between people. The invention of smartphones revolutionized the mobile phone industry while data sharing took another leap, both with respect to the used technology and the sheer scale of exchanged data. It is quite safe

to predict that the next large leap in this direction will come when embedded systems (ES) start to interact by exchanging data and collaborating towards common goals.

While today most ESs are developed for a particular application and operate on limited sets of sensor and actuator signals, interacting ESs will have a much wider choice of signals to use, offering vast opportunities for new emergent services. An example of such a service is a traffic intersection management system that collects data from the approaching vehicles and transforms that data into control signals for the vehicle speed, achieving a smooth traffic flow through the intersection. Numerous other examples of emergent services, either real or imagined, can be found in many different application domains, such as automotive, transportation, construction, healthcare, manufacturing, energy, etc. [2, 4, 13, 19]. This has spawned several interesting and somewhat related research directions, such as cyber-physical systems, internet of things, systems of systems, ubiquitous systems, etc., each focusing on slightly different aspects of the concept. For a more detailed literature review, see [12].

In our work, we use the term federated embedded systems (FES) to emphasize the focus on embedded systems and the concepts that are needed in order for ESs to be able to interact with each other in a meaningful way. The interactions are modeled as federations of systems, both embedded and traditional, where each system in some way benefits from participation in the federation. The FESs may either be static or evolve dynamically, both with respect to their functionality and composition. In many ways, the term FES is related to the field of cyber-physical systems, but is more focused on the concepts needed for the creation and operation of federations.

To reach the FES vision, significant advances in several research directions are needed. This includes mechanisms to dynamically join, operate in, and leave federations, as well as methods for handling security, software and hardware faults, conflicting requirements, information modeling, software architecture, privacy issues, embedded systems technology, and others. While the emergent FES functionality should bring some benefits to all participating ESs, the individual ES functionality, especially the safety critical one, must be maintained. Also, the concepts need to prepare for the FESs being open, both in the sense of openness towards new FES members and in terms of open innovation, with third party developers providing software to the ESs that would enable them to participate in a specific federation. Thus, new business models will be needed that support new types of software ecosystems.

Obviously, the challenges are numerous. To get a better understanding of the FES concepts and challenges, we conducted a pre-study on the FES subject, based on a series of workshops together with several industrial partners [13]. A portfolio of applications from different application domains was collected and used as the basis for the discussions. It became evident that in order to reach the FES vision, both technological and methodological advances are needed.

The main point of this paper is to summarize the concepts of our pre-study, and to present concrete work towards the FES vision that followed. In Section 2, some high level concepts of FES are presented. Section 3 describes a software component concept that enables dynamic software reconfiguration during runtime in vehicle applications. Section 4 presents our work within software ecosystem methodology, Section 5 reviews some related work, while Section 6 concludes the paper.

2 High level concepts

In this section, main FES-related concepts that were put forward during our prestudy [13] are summarized. Basically, the concepts were partitioned into four groups, divided by two conceptual axes, technology vs. methodology and system-level vs federation-level concepts. In

the following subsections, these concepts are shortly presented.

2.1 System-level Technology Concepts

On the level of individual systems, some basic technologies are needed in order for the ESs to be able to participate in federations with other systems. First of all, in order for the federations to form and for the ESs to contribute to and benefit from the FESs, the systems must be able to communicate with each other. Thus, technology for external communication is needed.

Secondly, federations and the services that they provide will often be evolvable and unforeseen at the design time of individual ESs. For this to happen, it should be possible to dynamically add and update software to the ESs at runtime. In consequence, if safety-critical functionality is allowed to be affected in such a way, there should be fault handling mechanisms that monitor how the new software complies with the system requirements, both functional and non-functional, and resort to inbuilt fall-back functionality if needed. Also, faults can be caused by the newly added software containing conflicts with other parts of installed software. Thus, logical software conflicts should be detected and handled.

In a recent work, a conceptual model for dynamically updatable embedded software was proposed [5]. It builds upon AUTOSAR [1], an architecture standard being widely used in the automotive sector. Currently, the concept is being further developed, in parallel with the development of tools and a demonstrator to show different FES application scenarios. The model is highlighted in Section 3.

2.2 Federation-level Technology Concepts

At the federation level, the technology needs are more intricate. Standardized protocols are needed in order for different kinds of ESs to cooperate. Such protocols should describe the communication details and the rules to which participating ESs will have to abide while functioning within a certain federation. In most federations, different types of ESs will play different roles, thus following different sets of rules.

New fault handling mechanisms are needed to handle the emergent behavior. On one hand, faults that would never exist in separated ESs may occur due to interactions. On the other hand, ESs may assist each other to overcome or to reduce the effects of faults. Again, faults can be caused by conflicting functionality. However, this time the level of abstraction is higher and the conflicts are expected to occur between ESs and their differering requirements. Related topics of importance are trust and uncertainty management in the scope of a federation.

2.3 System-level Methodology Concepts

In order for the FES to become a reality, methodology related concepts must not be neglected. Today, an ES is generally produced by one original equipment manufacturer (OEM), as part of a larger product. It often contains parts, both hardware and software, from different suppliers, while the OEM is responsible for integration.

With the idea of dynamic software, the number of participating software producers will be even higher, and since third-party developers will be able to add software without the involvement of the OEM, roles and responsibilities change between the parties. This, in turn, will change information flows during development and affect tools. A successful ES will no longer be one that only provides a certain function, but one that serves as a useful platform for adding new functionality on top of it.

2.4 Federation-level Methodology Concepts

While interactions between different actors that contribute to ES development may be complex, they become even more entangled at the federation level. To pave the ground for evolving and persistent federations, well defined business models are crucial. On the one hand, such models should provide opportunities for different parties to benefit from the emergent functionality, encouraging them to participate in the operation and development of the federation. On the other hand, the responsibility for the federation should be clearly defined. In other words, all aspects of the emergent functionality should be owned and maintained by some stakeholder.

The distribution of responsibilities and benefits between stakeholders is a challenging question. Even more challenging is how to do this dynamically in order to keep up with the changing nature of FES. The solution should include possibilities to allow new parties to take part in the federation operation, to let existing parties to take on new roles if needed, and to adapt responsibilities to the evolving FES functionality.

It seems clear that the technological development in itself is not sufficient for the creation and evolution of lasting FESs. The methodological aspects of federation operation should be carefully investigated. In Section 4, our initial work on this subject is presented.

3 Dynamic Software Reconfiguration in Embedded Systems

In this Section, a component model that allows dynamically adding and removing parts of ES software is presented. This model is a concrete example of a system level enabling technology that opens up for third party developers to add new services to ESs, ultimately creating opportunities for FES formation. The model is primarily tailored for automotive applications and builds on the AUTomotive Open System ARchitecture (AUTOSAR) standard [1]. However, the standard is not limited to the automotive world. In fact, it is suitable to all ES applications where the basic software (e.g. task scheduler, device drivers, hardware abstractions, etc.) is common to several control units and can be standardized.

In the following subsections, the AUTOSAR architecture is briefly introduced, followed by our extensions to the concept together with a few implementation details. Finally, some safety and security related remarks are collected.

3.1 The AUTOSAR Concept

AUTOSAR is structured around a layered software architecture that decouples the basic software (BSW) from the application software (ASW). This is accomplished by means of a component model, and a middleware called the runtime environment (RTE). Using AUTOSAR, ASW is modeled as a collection of software components (SW-C), which are in many ways similar to established component models like Koala [22], that communicate with each other and the rest of the system (e.g. standardized BSW) through so called ports. The internal functionality of the component only accesses its ports.

The actual communication between the ASW components, as well as their access to the lower layers, is taken care of by the RTE by interconnecting appropriate ports. This eases reuse of parts of the ASW, while RTE adds flexibility and scalability to the AUTOSAR architecture, allowing application SW-Cs to be easily redistributed between different control units simply by reconfiguring the RTE.

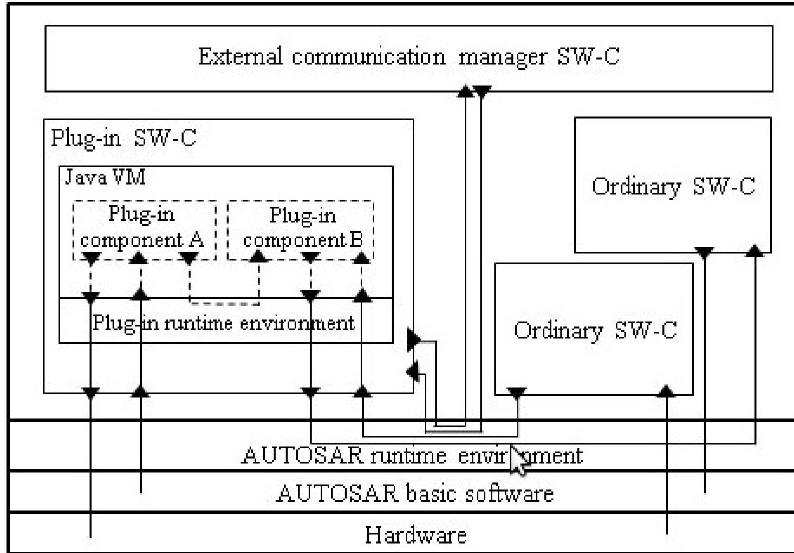

Figure 1: The structure of a dynamic software component model.

However, AUTOSAR has been designed to execute with limited resources and hence configuration of the system, such as allocation of SW-Cs to control units, and connection between SW-C ports, is done at design time with no structural dynamics during execution. The configuration is described in xml-files separate form the source code. These description files are used before deployment to generate C code that links ASW to BSW. Any changes in configuration require the software to be rebuilt and the control unit to be reprogrammed.

3.2 Dynamic Software Components

In [5], initial work on a conceptual model that extends the AUTOSAR architecture to allow software update at runtime was proposed. The key is to extend the set of ordinary application SW-Cs with dedicated SW-Cs for running additional software, hereafter called plug-in software, which is installed after the vehicle has left the factory. In this work, only plug-in enabling concepts are presented, while the internal plug-in functionality, which actually defines the rules for how the ES may act on the federation level, see Section 2.2, is not considered at this point, but will be addressed in the future.

Figure 1 gives an overview of how the plug-in concept relates to the underlying AUTOSAR based software. In the figure, dotted lines are used to show the plug-ins and their connections, whereas solid lines are used for the AUTOSAR SW-Cs and their links. For the concept to work, the OEM must provide plug-in enabled SW-Cs, which to start with only contain a Java virtual machine (VM) and an API that will be available to the plug-ins in the form of input and output ports, connected to the rest of the system through AUTOSAR RTE.

Also, one external communication manager (ECM) SW-C is needed, capable of communicating with a pre-defined external trusted server so that plug-ins can be installed, updated, and uninstalled at runtime. Furthermore, ECM serves as a gateway for plug-ins to communicate externally, which allows transferring information to and from off-board services, and participating in FESs. Finally, the AUTOSAR RTE must be configured so that ECM is connected to

the plug-in SW-Cs.

3.3 Internal communication

Inside the plug-in SW-Cs, AUTOSAR concepts are replicated as far as possible. Plug-in components communicate with the rest of the system through ports, while the connection details are configured in the plug-in runtime environment (PIRTE). Differently from the AUTOSAR RTE, the PIRTE contains both a static and a dynamic part. The static PIRTE part interfaces with SW-C ports and maps them into Java API signals. The dynamic part, updated each time any plug-in SW-C is updated, handles plug-in ports and their connections.

Plug-in ports can either access built-in functionality through the API provided by PIRTE, or they can be connected to ports on other plug-ins, again mediated by PIRTE. This is done even if the plug-ins are part of the same application, allowing dynamic reallocation of plug-ins between control units if needed. For example, if plug-in B in Figure 1 were reallocated, PIRTE would pass the connection to the AUTOSAR RTE that in its turn would forward it through the databus to the correct control unit.

Since it is not practically possible for the OEM to provide (and connect) SW-C ports for all imaginable future plug-in ports, the communication between plug-ins on different control units is done through dedicated SW-C ports (one pair of ports per control unit), which are fully connected to each other in AUTOSAR RTE. As a result, PIRTE needs to provide the address of the receiving control unit, the receiving plug-in, and the receiving port in that plug-in with the message that is passed to the data bus. Note that all these communication details only affect PIRTE and are transparent to the plug-ins.

3.4 Safety & Security

To provide a basic level of security, plugin software is sandboxed in as far as possible. First of all, plug-ins can only access the underlying system through the ports of the plug-in SW-C. It is up to the ECU developers to decide which ports to provide and how data received from these ports should be handled. If that data is used to control the underlying system, it is important that (non-reconfigurable) fallback mechanisms, with the authority to override plug-in actions, are in place. Secondly, Java VM executes in its own thread and with its own memory areas and network messages. This avoids competition for resources with the built-in functionality. Plug-ins are thus executed under a best effort scheme, whereas built-in software has predictable behavior.

A potential security threat is the installation of plug-ins. In this concept, it is only allowed to install plug-ins from a trusted server at a pre-defined address. In this way, much of the firewall issues are moved from the resource-constrained embedded system to a server. To change the trusted server address requires reprogramming of the ECUs built-in software, which has its own security mechanisms.

4 Ecosystems of embedded systems

In order to create a successful concept for FES, it is not sufficient to only look at the technical implementation, but one must also study how to organize development of the systems and to achieve sustainable business models, as described in Section 2.4. The key concepts of FES actually provide several opportunities from a business perspective:

- The plug-in components can be used by an OEM to add new functionality very rapidly, thereby being more responsive to market trends or to requirements from niche users.
- Third-parties can develop plug-in components to extend the functionality beyond what the OEMs conceived, similarly to how app developers extend the functionality of mobile phones.
- Systems can be integrated into systems-of-systems, whose functions are realized by distributed software. The integration in this case is handled by a separate organization.

This means that many stakeholders have an interest in the development and use of a FES, and the interrelations between them become crucial. In traditional development of ES, there is an OEM who is responsible for the end product, and who integrates subsystems from different suppliers. The suppliers develop their parts based on detailed specifications from the OEM.

In the FES setting, the OEM instead delivers an extensible product, whose success is partly a result of how well it supports the independent development of add-on solutions, and not only how well it meets the basic requirements. Based on this platform, a thriving business ecosystem [7] can be created, where third-party actors can practice open innovation and thereby extend the value of the base product.

However, this also leads to new challenges that need to be addressed when it comes to system development. For instance, ways of sharing information between different parties must be found, so that a plug-in developer can develop and test its software without full access to the overall product. Quality assurance in general is an issue, and the base product has to be tested with respect to all possible plug-ins that can be added to it. The distribution of rights and responsibilities between the parties are also crucial. Who is liable in a situation where an incident occurs? How should the streams of income be set up and divided among the parties?

Clearly, the business side and the technical side of FES are not separate. They meet in, for instance, the product architecture, that must support in a good way the development of both base products and plug-ins, and continue to do so over the time that the system evolves.

To investigate these issues, we are currently conducting an empirical research project, where we, based on case studies and interviews, try to identify the primary interfaces between stakeholders in the ecosystem. This will be used to create a reference model that explains what flows of information, money, etc. exist between them, and can be used to provide guidance on how to organize the ecosystem efficiently [16].

5 Related work

Several studies on the subject of cooperating systems have been published. In [6], more than 40 different definitions of the term systems of systems were reviewed and classified, while the notion of federations of systems was coined in [20]. In [4], recent research advances within the internet of things field, together with a number of application scenarios, are presented. In the field of cyber-physical systems, a lot of work has been done to present both the research challenges and possible applications, e.g. [2, 14, 21, 18]. A more extensive litterature review can be found in [12].

While all the above research directions are interrelated, they focus on somewhat different questions, even though they seem to be starting to merge. For example, the IoT field has sprung out of the desire to be able to uniquely identify any physical object, with a large focus on identification and communication. The SoS research on its hand has traditionally been focusing on the engineering management studies. The CPS field seems to be the most related

to the FES concept. In fact, one could argue that CPS actually is a larger field containing FES. The difference lies in the strong emphasis that the term FES places on the use of embedded systems in federations that are generally allowed to be dynamically reconfigurable, both with respect to their composition and functionality. The perceived need for a focus on open and dynamic federations of ESs was one of the reasons for our FES pre-study [13]. Another reason was to give the opportunity for the Swedish industry to add its voice to the ongoing evolution of the CPS concept, zooming in on FES related challenges.

When it comes to dynamic reconfiguration of SW-Cs, it has been studied in e.g. [3, 8, 17, 23]. Differently from the above publications, this work provides a Java-based and simple to implement concept that builds on a standardized architecture, offering good opportunities for open software development.

The term "ecosystem" was introduced by Iansiti and Levien who described the notion of business ecosystems [10]. They explained the benefits of adopting the "ecosystem-thinking" from a business perspective and discussed various strategies organizations may utilize, depending on their role within the ecosystem. However, the article does not explicitly describe how to carry out product development, or what specific characteristics products should have in order to make the best out of such an ecosystem.

The term "software ecosystem" was introduced by Messerschmidt and Szyperski [15] but was extended by Bosch [7]. In particular, Bosch extended the classical "product line-thinking" of software products, The trend towards open platforms was started because it is too expensive for an manufacturer to develop alone all the functionality that customers would wish for, and because gathering the requirements for customization could potentially be done more efficiently through an open platform.

Hanssen and Dyb [9] described in their work a systematic overview of software ecosystems and explained several related challenges. Jansen, Finkelstein and Brinkkemper [11] presented a research agenda for software ecosystems, discussed about the main challenges involved in a technical and business level through three dimensions: a) from a software ecosystem level, b) software supply network level, and c) from the software vendor level, and also mentioned issues of formal modeling, transparency, guidelines, standards, and actions, that are of central importance.

All in all, there is very little research that looks at ecosystems specifically for ES, but the literature either is looking on pure software, or general product development.

6 Conclusions and future work

This paper consists of three main parts. Firstly, high level concepts related to FES and its constituent ESs are presented. These concepts are further divided into those that are connected to technology development and those that relate to product and process development methodologies, e.g. business models. Secondly, our work in the technological direction is presented. This work extends the AUTOSAR architecture with the concept of dynamic component models, thus allowing installation of new plug-in software into vehicles at runtime, opening up for implementing FES governing interaction rules long after the vehicles have left factory. Although the AUTOSAR standard is from the automotive industry, the concepts are quite general and of value in other embedded system domains. Thirdly, we present our initial work in the methodology direction, aiming at defining the business models necessary for dynamic FES ecosystems.

While our work on FES methodology is currently just taking off, see Section 4 for a discussion of future challenges, there is more to say about the continuation in the technological direction. More realistic tests of the concepts presented in Section 3 are needed to get deeper insights about

the strengths and weaknesses of the proposed solution. Stressing the system in order to test robustness is also important. For example, this could help to understand what happens if power goes off during the installation of a plug-in, or how to react to the loss of messages between plug-ins, etc. To increase the practical usefulness of the proposed architecture, tools that aid in generating plug-in runtime environment will be developed. Also, in theory the concepts should be applicable to safety critical applications. However, for this to work, there is a need of fault handling mechanisms, both locally at the ES level, provided by the ES developers, and at the FES level. Trust and conflict management mechanisms are other intricate but interesting research questions. Once the basic concepts and the simulation environment are in place, such more complex issues are ready to be addressed.

6.1 Acknowledgements

The projects presented in this paper are supported by Vinnova (grants no. 2012-02004 and 2012-03782), Volvo Cars, and the Volvo Group.

References

[1] AUTOSAR consortium. Available: http://www.autosar.org/. Last accessed 29 Jul, 2013.

[2] CPS summit report, 2008. Available: http://varma.ece.cmu.edu/Summit/CPS_Summit_Report.pdf. Last accessed 29 Jul, 2013.

[3] Richard Anthony, Achim Rettberg, Dejiu Chen, Isabell Jahnich, Gerrit de Boer, and Cecilia Ekelin. Towards a dynamically reconfigurable automotive control system architecture. In *Embedded System Design: Topics, Techniques and Trends*, pages 71–84. 2007.

[4] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer Networks*, 54(15):2787–2805, 2010.

[5] Jakob Axelsson and Avenir Kobetski. On the conceptual design of a dynamic component model for reconfigurable autosar systems. In *Proceedings of the Workshop on Adaptive and Reconfigurable Embedded Systems (APRES)*, 2013.

[6] John Boardman, Spiros Pallas, BJ Sauser, and D Verma. Report on system of systems engineering. Technical report, Final Report for the Office of Secretary of Defense, Stevens Institute of Technology, Hoboken, NJ, 2006.

[7] Jan Bosch. From software product lines to software ecosystems. In *Proceedings of the 13th International Software Product Line Conference*, pages 111–119. Carnegie Mellon University, 2009.

[8] Meik Felser, Rüdiger Kapitza, Jürgen Kleinöder, and Wolfgang Schröder-Preikschat. Dynamic software update of resource-constrained distributed embedded systems. In *Embedded System Design: Topics, Techniques and Trends*, pages 387–400. 2007.

[9] Geir K Hanssen and Tore Dybå. Theoretical foundations of software ecosystems. *Proceedings of the 4th Software Ecosystems Workshop (IWECO)*, pages 6–17, 2012.

[10] Marco Iansiti and Roy Levien. Strategy as ecology. *Harvard business review*, 82(3):68–81, 2004.

[11] Slinger Jansen, Anthony Finkelstein, and Sjaak Brinkkemper. A sense of community: A research agenda for software ecosystems. In *Proceedings of the 31st International Conference on Software Engineering*, pages 187–190, 2009.

[12] Avenir Kobetski and Jakob Axelsson. Federated embedded systems ? a review of the literature in related fields. Technical report, Swedish Institute of Computer Science, 2012.

[13] Avenir Kobetski and Jakob Axelsson. Federated robust embedded systems: Concepts and challenges. Technical report, Swedish Institute of Computer Science, 2012.

[14] Edward A Lee. Cyber physical systems: Design challenges. In *Proccedings of the 11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing (ISORC)*, pages 363–369, 2008.

[15] David G Messerschmitt and Clemens Szyperski. Software ecosystem: understanding an indispensable technology and industry. *MIT Press Books*, 1, 2005.

[16] Efi Papatheocharous, Jakob Axelsson, and Jesper Andrersson. Issues and challenges in ecosystems for federated embedded systems. In *Proceedings of the First International Workshop on Software Engineering for Systems-of-Systems*, SESoS '13, 2013.

[17] Juraj Polakovic, Sebastien Mazare, Jean-Bernard Stefani, and Pierre-Charles David. Experience with safe dynamic reconfigurations in component-based embedded systems. In *Component-Based Software Engineering*, pages 242–257. 2007.

[18] Ragunathan Raj Rajkumar, Insup Lee, Lui Sha, and John Stankovic. Cyber-physical systems: the next computing revolution. In *Proceedings of the 47th Design Automation Conference*, pages 731–736, 2010.

[19] David Rylander and Jakob Axelsson. Using wireless communication to improve road safety and quality of service at road construction work sites. In *IEEE Vehicular Networking Conference (VNC)*, 2012.

[20] Andrew P Sage and Christopher D Cuppan. On the systems engineering and management of systems of systems and federations of systems. *Information, Knowledge, Systems Management*, 2(4):325–345, 2001.

[21] Lui Sha, Sathish Gopalakrishnan, Xue Liu, and Qixin Wang. Cyber-physical systems: A new frontier. In *Machine Learning in Cyber Trust*, pages 3–13. 2009.

[22] Rob Van Ommering, Frank Van Der Linden, Jeff Kramer, and Jeff Magee. The koala component model for consumer electronics software. *Computer*, 33(3):78–85, 2000.

[23] Yves Vandewoude and Yolande Berbers. Run-time evolution for embedded component-oriented systems. In *Proceedings of the 18th International Conference on Software Maintenance*, pages 242–245, 2002.

A Roadmap Towards Integrated CPS Development Environments

Jad El-khoury, Fredrik Asplund, Matthias Biehl, Frederic Loiret and Martin Törngren

Mechatronics Lab, Department of Machine Design, Royal Institute of Technology, Stockholm, Sweden
{jad, fasplund, biehl, floiret, martint}@kth.se

Abstract

Cyber Physical System (CPS) development is highly heterogeneous, involving many stakeholders, each of which interacts with its development artifacts through a variety of tools, and within several engineering processes. Successful CPS development requires these tools to be well-integrated into a Development Environment (DE) in order to support its many stakeholders and processes. In this paper we identify the main challenges facing DE development for CPSs, and presents a roadmap to meet these challenges. We here take the position that focus should be redirected from trying to achieve a single, one-size-fits-all solution to such a heterogeneous problem. Instead, focus should be placed on supporting the development of highly-customized DEs, which readily can be applied to industrial development. Such a highly-customized DE should fit the needs of a particular development organization, while at the same time taking advantage of relevant standardization efforts.

1 Introduction

Cyber Physical Systems (CPSs) offer opportunities for new services, improved performance and better efficiency in almost all application domains in our society.

Typically many technical and business-related stakeholders take part in the development of a CPS. Each of these stakeholders interacts with the development artifacts through various tools, pertaining to their particular interests. Mechanical engineers for instance make use of CAD tools to construct artifacts that describe the mechanical aspects of the CPS under development. This interaction takes place within several technical engineering processes involving structured sequences of activities such as requirements engineering, design and verification.

Due to the tightly integrated technologies in a CPS, all of these technical engineering processes become tightly intertwined and decisions made by one stakeholder become likely to have an impact on other stakeholders. An effect of this is that the tools supporting each separate technical engineering process needs to be adequately integrated with tools of other technical engineering processes. This becomes problematic, since many of the tools employed throughout the different processes typically come from separate sources and are hence likely to be mutually incompatible.

This paper focuses on the overall problem aimed at supporting the integration of these mutually incompatible tools. We will use the term Development Environment (DE) to refer to a setting of tools that support multiple stakeholders and processes in CPS development. How to design and maintain DEs have been discussed thoroughly these last three decades, with the overall focus being on trying to achieve a single, one-size-fits-all solution towards which all mutually incompatible tool technologies subsequently can interface (thereby forming a homogeneous unity).

We believe that the current approaches to building DEs are built on a false hope that CPS development can be homogenized, so that a single homogeneous DE integration framework to support the whole CPS development life-cycle can be provided. As we will argue in this paper, CPS development is too heterogeneous to allow for such a single homogeneous solution. Instead, we accept the heterogeneous nature of CPSs and their DEs, and aim to instead provide well-integrated heterogeneous DEs.

1.1 Position Statement

Given the heterogeneous nature of CPS development, and the (current) slow pace of convergence of integration technologies, we believe that focus should be redirected at supporting the development of highly-customized and maintainable DEs, which readily can be applied to industrial development. Such a highly-customized DE should fit the needs of a particular development organization, while at the same time taking advantage of relevant standardization efforts.

A tailorable organization-specific DE can only be practically feasible if the threshold of its development is lowered by providing DE development and automation support. Moreover, a concerted effort will be required to provide the required methodology, standards and business models required for large scale industrial adoption of efficient tool integration.

1.2 Paper Structure

This paper first describes in section 2 the heterogeneous context of CPS development, leading to the main challenges of developing DEs - which are elaborated in section 3. Section 4 finally summarizes the issues that need to be addressed to move forward and some of the opportunities that this will lead to.

2 The Heterogeneous Context of CPS Development

The CPS development process is highly heterogeneous. In this section we go through some of the more important aspects in which this has an impact on DEs.

2.1 Tool and Tool Integration Heterogeneity

As mentioned in section 1, many stakeholders with different technical and business-related specialities mean that many heterogeneous tools will be found in a DE. The tightly intertwined technologies in an CPS will then mean that many tools will be integrated with each other, forming complex dependencies between them.

A DE is often built in a bottom-up manner, eventually displaying an unstructured design and implementation of tool integration. Such ad-hoc realizations of DEs may use a variety of integration frameworks, data formats, communication protocols and assumptions. Integration conventions provide a common ground for building DEs and increase the likelihood that parts of DEs can be reused in a different context than they were originally designed for. However, several conventions for integration exist, such as XMI (XML Metadata Interchange) (OMG, 2007), OSLC (Open Services for Lifecycle Collaboration) (OSLC Core Specification Workgroup, 2010) and STEP (ISO, 1994). Similarly, several specialized technologies for realizing the different parts of a DE are currently available, such as model transformation tools, tracing tools or libraries for exposing services of tools. Each of these integration technologies describes only one aspect of the DE, while a complete DE needs to cover several aspects.

This plethora of tools, integration frameworks, integration conventions, languages and technologies for realizing parts of a DE, combined with the common ad-hoc realization approach, typically lead to a rich heterogeneity of technologies used for tool integration.

2.2 Organizational Heterogeneity

Organizations have different development processes of varying technical maturity. For example, while a more traditional DE supports simple connections between a small number of tools; a modern DE may need to support development processes that are model-based and iterative (Tratt, 2005) and include a larger number of tools.

In addition, to get a DE accepted, it is important that its end-users (such as the different types of engineers, architects, managers, designers, analysts, etc.) are involved (Christie et al., 1997). DEs therefore need to be tailored to each specific organization, or even each specific development project. As a consequence, the requirements and nature of tool integration vary among organizations.

2.3 Stakeholder Heterogeneity

The DE end-users frequently see the ideal case as being when they can focus on one tool to support a particular ”main” activity and then have information automatically flow to and from this tool (Maalej, 2009). However, DE users are not the only stakeholders relevant to tool integration.

DE designers, deployers and maintainers have a different view of each tool and the overall DE. They instead usually favor tampering as little as possible with each tool or technology employed for tool integration.

And while those stakeholders have a common ground in the focus on technology, other stakeholders have an all together different focus. Management commonly looks at the *cost* of procuring technologies and tools when deciding which solution to favor. Avoiding a potentially costly ”lock-in” in regard to a particular technology can lead to the rejection of tools that are technologically superior.

This focus on economical factors can also be found in the reasoning of tool vendors, which may be interested in tool integration as an argument for customers to choose their tools (for instance to increase the odds of avoiding a costly ”lock-in”). However, tool vendors with market unique or dominating solutions have less of a reason to offer support for much tool integration technologies. This support comes with a cost, both during development and maintenance, which a tool vendor might want to avoid since it is not their main business.

3 The Challenges of DE development for CPS

Given the heterogeneous nature of CPS development, we here identify and elaborate on the most important challenges facing DE development.

3.1 Lack of Support Methodologies and Tools

A DE consists of many distributed software assets to be integrated with each other. This is complicated due to the heterogeneity described in section 2, and the many intricate dependencies mentioned in section 1. Several barriers have to be bridged, including *technology* (e.g. technical representation of information and functionalities), *semantics* (meaning of information and functionalities, and their relations), and *intended interactions* (scenarios involving two or more tools).

Part of the problem is the lack of an established methodology for the development of DEs, meaning that DEs are often implemented in an ad-hoc manner. This leads to “fragile integrations” (Derler et al., 2012) that are difficult to extend and maintain.

Another level of complexity is introduced when needs for customization have to be considered, e.g., to take into account product-specific lifecycle features and integration patterns.

Current platforms for tool integration provide partial solutions, but also introduce accidental complexity through the amount of manual coding, low-level technologies and configurations. The lack of support methodologies comes along with a corresponding lack of support tools that would offload the burden on DE developers and integrators at various stages of the development process. For instance, high level modeling languages for designing and supporting early testing of DEs, coupled with adaptive composition mechanisms and automatic synthesis of integration assets, are typical support tools that are not available to enrich the portfolio of DE developers and integrators.

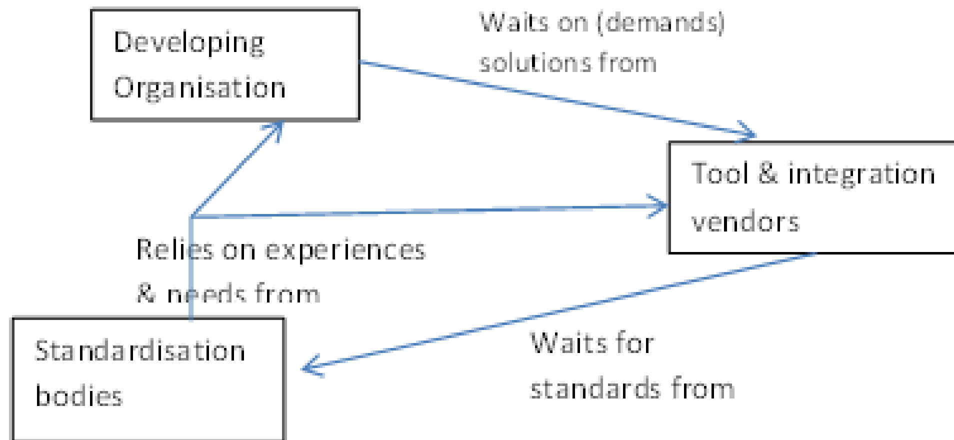

Figure 1: key stakeholders integration technologies and their dependencies.

3.2 Convergence towards New Standards for Tool Integration

Supporting methodologies and tools needs to be complemented by suitable standards. The question is, however, how come there is so few widely adopted standards for tool integration, when there is clearly no lack of suggestions for how to achieve standardization? There is for instance a multitude of suggestions for data exchange formats (such as XMI (OMG, 2013b), FMI (FMI Development Group, 2013) and ReqIF (OMG, 2013a)), modeling languages (such as EAST-ADL (EAST-ADL Association, 2013)) and even complete frameworks (Such as Jazz (IBM, 2013) and ModelBus (ModelBus Team, 2013)) to support one or more aspects of tool integration.

It could have been expected that - over time - momentum would have been picked up behind a selected few of these diverse suggestions, leading to a more concentrated effort towards a settled set of basic standards. Time and experience is after all needed for this kind of maturity process, given the many stakeholders (and their complex relationships) involved in any such efforts. While this process is natural for any emerging technology, the heterogeneous context of CPS development (as presented in section 2) unfortunately causes a prolongation of the time period until such stable standards are in place - if at all.

Furthermore, as mentioned in section 2, tool integration is a secondary objective for many of the key stakeholders. For example, CPS developers and the business units they belong to want to develop a product, while tool vendors want to develop tools with minimum effort and sometimes believe that open tools is a threat rather than opportunity, etc. In the best case, business units can cooperate with such vendors to solve their particular integration needs, but it leads to organization-specific (and in many cases ad-hoc) integration solutions. This obviously further prolongs the time until standards pick up enough momentum to become widely accepted.

Figure 1 illustrates the cyclic nature of the challenge in engaging the key stakeholders through constructive interactions in order to converge on integration technologies. The lack of support methodologies and tools contribute to the vicious nature of the cycle.

4 Concluding Discussion

We believe there is a need for a new approach in order to break the vicious circle illustrated in Figure 1. As mentioned in section 3, the key stakeholders lack the incentive to push for a solution to such a

need, given their own primary objectives.

A very important aspect is to raise the level of maturity and awareness of the developing organizations in order for them to be able to take a more active role in contributing to workable standards.

Another important aspect is to establish integration methodologies and standards that take the heterogeneity and needs for customization explicitly into account. This will enable tool vendors to provide customized integration solutions that can be relatively easily and cheaply provided for any developing organization.

Finally, there is an opportunity for additional business players to provide standardized tool interfaces for existing development tools and use these interfaces to create customized and automated development environments. We call these business players *integration providers*. A business model for such a business player would take advantage of the current vacuum to provide customized, standardized tool integration, allowing these players to act as product or service providers (See (Murphy and Duggan, 2012) for example).

Maturing methodologies and standards will strengthen the opportunities of integration providers, promising to create a successful industrial ecosystem. Getting there will require both research and collaborative efforts among key stakeholders.

REFERENCES

Alan Christie, Linda Levine, Edwin J. Morris, Bill Riddle, and David Zubrow. Software Process Automation: Interviews, Survey, and Workshop Results. Technical report, SEI, 1997.

P. Derler, E. A. Lee, and A. S. Vincentelli. Modeling Cyber-Physical Systems. *Proceedings of the IEEE*, 100(1):13–28, January 2012.

EAST-ADL Association. East-adl, 04 2013.

FMI Development Group. Functional mock-up interface, 04 2013.

IBM. Ibm rational jazz, 04 2013.

ISO. Industrial automation systems and integration – product data representation and exchange (ISO 10303). Technical report, ISO, 1994.

Walid Maalej. Task-First or Context-First? Tool Integration Revisited. In *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*, ASE '09, pages 344–355, Washington, DC, USA, 2009. IEEE Computer Society.

ModelBus Team. Modelbus, 04 2013.

Thomas E. Murphy and Jim Duggan. Magic quadrant for application life cycle management. Technical report, Gartner, 2012.

OMG. MOF 2.0 / XMI Mapping Specification, v2.1.1. Technical report, OMG, December 2007.

OMG. Requirements interchange format (reqif), 04 2013.

OMG. Xml metadata interchange, 04 2013.

OSLC Core Specification Workgroup. OSLC core specification version 2.0. Technical report, Open Services for Lifecycle Collaboration, August 2010.

Laurence Tratt. Model transformations and tool integration. *Software and Systems Modeling*, 4(2):112–122, May 2005.

Engineering of Cyber-Physical Systems

María Victoria Cengarle[*]

fortiss GmbH, München, Germany
cengarle@fortiss.org

Abstract

The present work reflects on the drivers and barriers of Cyber-Physical Systems with focus on the challenges associated with their engineering. Based on a previous survey, the charachteristics are studied that a reference framework for the engineering of Cyber-Physical Systems should support. Thereby orthogonal dimensions of the envisioned systems and proposed system development phases, on the one hand, and diverse solutions in use, on the other, are collated.

1 Introduction

The present work refers insights and thoughts concerning the mastering of Cyber-Physical Systems (CPS). Nowadays CPS are touted as the next revolution in Computer Science. Forerunners can already be found in as dissimilar areas as automotive, avionics, energy, health, environmentalism and consumer electronics. The vision poses extraordinary challenges particularly regarding technology, organisation and human-system cooperation. It also entails a huge potential both for economy as well as for tackling problems of modern society.

Here we focus on the engineering challenges posed by CPS, and draft some raw ideas on how to meet those. We firstly introduce in Sect. 2 our definition of CPS. Secondly in Sect. 3, we perform a kind of meta-requirements analysis in order to find out the demands a reference framework for CPS must fulfil. Afterwards, in Sect. 4 we detail candidate methods for the different phases in which we divide the development process of CPS. Finally in Sect. 5 we draw some conclusions and outline a number of possible improvements of the currently available loose ends.

2 Cyber-Physical Systems

As defined in [4], a Cyber-Physical System (CPS)[1] is a system with embedded software (as part of devices, buildings, means of transport, transport routes, production systems, medical processes, logistic processes, coordination processes and management processes), which:

- directly records physical data using sensors and affect physical processes using actuators;
- evaluates and saves recorded data, and actively or reactively interacts both with the physical and digital world;
- is connected with other CPS and in global networks via digital communication facilities (wireless and/or wired, local and/or global);
- uses globally available data and services;
- has a series of dedicated, multimodal human-machine interfaces.

[*]This work has been partially sponsored by the EIT ICT Labs, project "Innovation Platform Cyber-Physical Systems Engineering".

[1]The term CPS is here used both as a singular and a plural noun, the number depending on the context.

The result of the connection of embedded systems with global networks is a wealth of far-reaching solutions and applications for all areas of our everyday life. Subsequently, innovative business options and models are developed on the basis of platforms and company networks. Here, the integration of the special features of embedded systems –for example, real-time requirements– with the characteristics of the internet, such as the openness of the systems, represents a particular technical challenge.

The main objective of the project "Innovation Platform Cyber-Physical Systems Engineering" is the integration, validation, and dissemination of a coherent, ready-to-use reference framework based on state-of-the-art science and technology as well as on the design and operation life cycle continuum of CPS Engineering (CPSE). It is also intented to instantiate and validate the CPSE by means of cross-domain scenarios and case studies. The long-term vision is to establish the EIT ICT Labs as the cross-domain, multidisciplinary open innovation platform for developing and maintaining a ready-to-use, open and standardised CPSE reference framework, that facilitates the transitioning of complex and trustworthy CPS to the marketplace.

Challenges and opportunities

The biggest challenge brought about by the engineering of CPS is the integration of (discrete as well as continuous) models and methods from different disciplines including not only technical ones like mechanical and electric/electronic engineering, computer science[2] and control theory but also ergonomics and human factors, economic ecosystems, social guidelines and legal stipulations. These "soft" aspects of CPS are crucial for the acceptance of CPS and therefore for their success.

In exchange, there are a number of very significant opportunities allowed for by CPS. Besides value creation and innovation, the most noticeable ones are the enhancement of accident prevention procedures, improved support of ageing population, and smart use of limited resources. These have been considerably emphasised in [4, 13].

Regarding only the computational discipline, on the one hand we have traditional Business Information Systems (BIS) and, on the other, traditional Embedded Systems (ES). The former are data-centric, ideally high secure and open, focus on maintenance, their life cycle incorporate legacy systems and evolves continuously, and their constraints fall in the category of weak real-time. Their engineering challenges are application integration, enhancement of running systems, re-engineering of legacy systems, and validation and prediction. The latter, on the contrary, are function-centric and closed, focus on construction, their life cycle consists of decommission followed by design, building and commission (i.e., legacy is not an issue), and their constraints fall in the category of hard real-time. Their engineering challenges are systems engineering (function, architecture, platform, and mechanics), safe function deployment, and verification; see [5] and also [9]. By CPS these both sorts of systems need be combined; considering their description above, it is redundant to stress that their reconciliation is anything but straightforward. Moreover, the large-scale dimension of CPS aggravates the situation.

Engineering discipline

Because of the considerations above, it becomes apparent that the Engineering of CPS (CPSE) calls for a radical new paradigm allowing the integrated construction, operation, adaptation and evolution of large-scale, long-living, heterogeneous, open, dependable (in particular, safe and secure), high-investment systems. There is a series of aspects to be considered for devising a new CPSE reference framework; see [5]. On the one hand, we have the continuous life cycle of CPS that amalgamates Integrated Devel-

[2] Computer science here encompasses in many cases only unsatisfactorily solved issues as, e.g., interoperability, adaptability and tailoring, learning, private data protection, fault tolerance, safety and security.

opment Environments (IDE)[3] and Operating Systems (OS)—and thus puts a combined functionality at disposal for the design, simulation and verification, deployment, operation, maintenance of CPS. In this context longevity, including self-documentation, self-reflection, self-adaptation and self-optimization, as well as criticality, i.e. uninterrupted operation modifiable at runtime, need be meaningfully provided for.

On the other hand, built-in support is necessary for dependability including safety and security ("BIS meets ES") as well as large-scale: built-in compositionality for construction and operation of CPS (thus confronting the larger-scale knot posed by CPS). Moreover, the envisioned CPS engineering must include online-models of system, environment and situation and of domain-views as well.

3 Demands on the engineering

Existing reference frameworks for, e.g., embedded systems and systems of systems do not address the new challenges posed by CPS: openness and heterogeneity, portability and interoperability across domain boundaries as well as situation awareness and self-evolvability, among others. However, some already existing networked embedded systems let the conjecture raise of the suitability of upgrading and aggiornamento of established embedded systems engineering frameworks.

Starting point for the development of a proposal is here an analysis of the demands to be observed by a suitable reference framework for the development of CPS. Such a framework has to reveal cross-cutting fundamental scientific and engineering principles that underpin the integration of cyber and physical elements across the addressed sectors. The tentative concepts of the CPSE reference framework include life-cycle processes, terminology, design principles, guidelines together with an adapted multiview framework.

Depending on the degree of tightness (cf. Fig. 1), cooperating systems can be viewed as different and cooperating CPS or as together composing a single (however big) CPS. The latter materialise in the case of diverse, even wide spread, but cooperating development teams; the former when, e.g., systems initially not meant to cooperate with each other are composed. The engineering issues vary accordingly. In the first case, we speak of "coarse-grained" aspects, while in the second we speak of "fine-grained" concerns.

3.1 On the coarse-grained level

Open and dynamic systems can be bundled in order to provide a service that may be realised by not a single but a chain of systems spontaneously cooperating. The thus arising systems' cooperation poses the implicit challenge of the identification of individual systems as well as the description of the service offered by these. What moreover means that an orthogonal modelling dimension is indispensable for dynamic and spontaneous cooperation.

As pointed out in [4, 13], the approach must rely, particularly during requirements analysis, on an interdisciplinary approach. CPS systems stem from most diverse domains and are operated by people about whose background almost no assumption can be made, thus ease of use of those combined systems is imperative. An orthogonal dimension of design, therefore, has to consider the presentation aspects of each of the systems as well as their tie points.[4]

The conjecture here is that these three design modelling activities, namely individual service(s) design, cooperation design and presentation design, be separated thus supporting modular and reusable

[3] IDEs are applications that facilitate efficient software development by providing not only a syntax-oriented source code editor but also build automation (compilation, linking, deployment, etc.) and debugging tools.

[4] Cooperation and presentation are akin to the navigational design and the presentational design web and hypermedia applications described in [16].

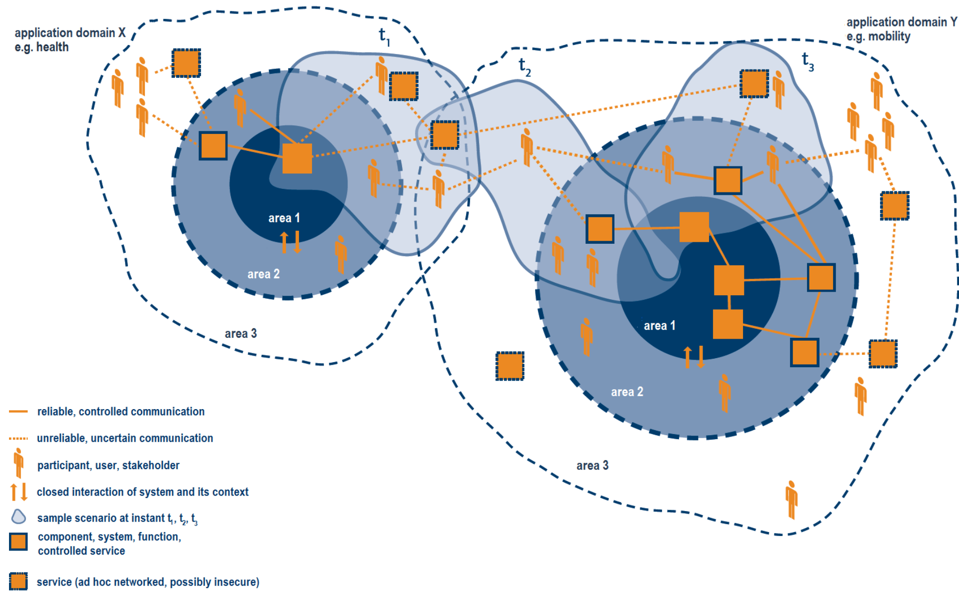

Figure 1: CPS domain structure (Source: [13])

design. Furthermore, the importance has been recognised of involving users in the innovation process, be they sources of innovative ideas, early testers in simulation environments, or even developers; see [2]. Living labs let researchers and engineers test and modify products in close collaboration with end-users in a real-life or a real-as-life setting. Living Labs capture users' insights, prototype and validate solutions, aim to contribute to both problems providing structure and governance to the user involvement and methodologies and organizations to filter and sense user insights; see [11]. Two prominent living labs are FutureEverything and the city of Oulu, Finland. FutureEverything is an art and digital innovation organization based in Manchester, England, around an annual festival of art, music and digital culture, that each year presents the work of around 300 artists across its art, music and conference strands, and is conceived as a living lab for participatory experiments on art, society and technology; see [15]. The world's first wristwatch rate monitor, GSM telephone call, WCDMA telephone call, etc. came from Oulu, whose dynamism is due to the Innovation and Marketing group of the city that acts as a Living Lab, setting up and analysing user experiences and laying out the service model; see [2]. Furthermore, in [21] the benefits are shown of children's participation in living labs.

3.2 On the fine-grained level

As already mentioned, it is unclear how conventional modelling techniques for BIS and for ES can be sensibly combined. One prominent problem profusely treated in the literature refers to modelling techniques addressing heterogeneity and hybridity (e.g., discrete vs. continuous models); see [10, 18, 24]. Also the integration of successful techniques for one realm into the other, for instance component-based engineering into ES design, as pointed out in [8] among others, is anything but obvious. In the microscopic level of embedded systems, dependability issues represent a non-trivial challenge that apparently cannot be enough warned of. This issue is magnified when one considers that, dynamic and spontaneously, services communicate and cooperate as an action or a reaction to the situation. Thus, mechan-

isms for authentication as well as for intention and need recognition ought to be improved/perfected or even devised where non-existent.

4 CPS Reference Framework

Aligned with the methodologies that were successful so far, we conceive a development methodology for CPS divided into phases that are to be composed and combined iteratively and successively and taking into account the different levels of refinement of single units as well as the different degrees of maturity of interacting systems. The purpose of this section is twofold: On the one hand, the above mentioned phases are seen from the perspective of CPS and, on the other, some specific (or dedicated) proposals and solutions nowadays in use are suggested that could cope with the task at hand.

The initial stage of any system is customarily called requirements analysis. Already for this first approach to a system there is in the realm of CPS (at least) two fundamentally orthogonal approaches. These are discussed in Sect. 4.1 below.

Reflecting on the further phases that can constitute a sound and all-encompassing reference framework for the development of CPS, we recognise different ways of approaching the challenges depending on the point of view assumed. The core application of a system (of systems) can be termed *service*, whose nature appears to be one from three possibilities: business, computation and control, or platform; see Sect. 4.2. The interplay between systems (of systems) requires the definition of rules for cooperation also comprising authentication, service description and communication protocols; see Sect. 4.3. Crucial for the acceptance of these by far non-obvious systems is the way they communicate with end-users, their ease (i.e., intuitiveness) of use, and their possible customisation; see Sect. 4.4.

4.1 Requirements Analysis

Similar to web applications, also CPS "facilitate business process integration, new business models, supply chain mediation disintermediation and reengineering, as well as offer new services to new markets"; see [19]. And likewise "potential users are so diverse and geographically wide-spread that [requirements analysis strategies predicated on consulting the future users of the systems] is impractical." In the cited work, thus, an approach to requirements elicitation is proposed that "combines the recognition of multiple user views of a complex human activity system with techniques to help creatively map existing and potential business functions to a Web-based environment [. . .] accessible to developers who are not IT function experts." This method, termed SSM/ICDT, combines the Soft Systems Methodology (SSM, see [22]) with the ICDT matrix (information, communication, distribution and transaction, see [3]). Because of the similarities mentioned, valuable insights may be gained by considering an activity-oriented approach to requirements analysis of CPS.

Alternatively, artefact-based approaches "promise to provide guidance in the creation of consistent artefacts in volatile project environments, because these approaches concentrate on the artefacts and their dependencies, instead of prescribing processes"; see [20]. The conducted a case study which showed the increased flexibility of the approach in comparison with the previously used one, as well as the improved quality of the created artefacts, and also that productiveness was not improved. A so-called mega-modelling environment termed Global Model Management (GMM, see [27]) permits typing, composition and execution of artefacts. As a consequence, type errors during execution can be avoided. This approach, appropriately transferred to the CPS setting, could used for authentication prior the establishment of a spontaneous cooperation.

4.2 Service(s) Design

It turns out that at least three kinds of services converge into CPS, that interact with each other. They are depicted in Fig. 2. An hypothesis worth considering is that the above difficulties of combining BIS and ES be solved by decoupling systems and let them only communicate via a platform (i.e., removing the dashed arrows in Fig. 2). This very probably implies a platform with considerably more intelligence than that of conventional ones.

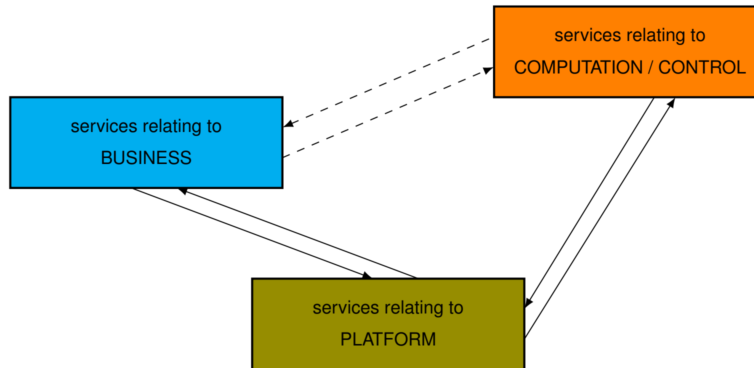

Figure 2: The three kinds of services

Much has been said and written about the immense costs of (fine) tuning, maintenance, and re-design and re-engineering of large scale complex IT systems. As stated in [23], "management structure that moves a megaproject along with seamless transitions between the project's phases can affect the final outcome and success".

Computation / control services

Because of its two dimensions of abstraction, the SPES Metamodel [14] seems an adequate starting point for the embedded systems dimension of service(s) modelling. On the one hand, there are the software development perspectives and, on the other, the levels of granularity; see Fig. 3. The former permit the examination of a system from different viewpoints and this way gain or specify diverse kinds of information about the system.

The functional perspective describes the systems from the angle of its usage. The logical perspective describes the system as a network of communicating and cooperating components possibly hierarchically structured. The technical perspective provides the technical details of the system especially with regards to hardware and virtual machine platform, and is conceived in such a way that it can be extended (or instantiated) for particular application domains.

The SPES Metamodel supports a process-based system development (see also [28]), into which the operation-design continuum of CPS may not fit smoothly. An alternative to be considered is a system development based, rather than on processes, on products; see [7].

Business services

Business services have been extensively addressed, at the beginning in manifold ways; see [1]. There are different approaches in this realm, so for instance Business Process Management (BPM), Service-Oriented Architecture (SOA), Service-Oriented Computing (SOC), etc.; see, e.g., [29].

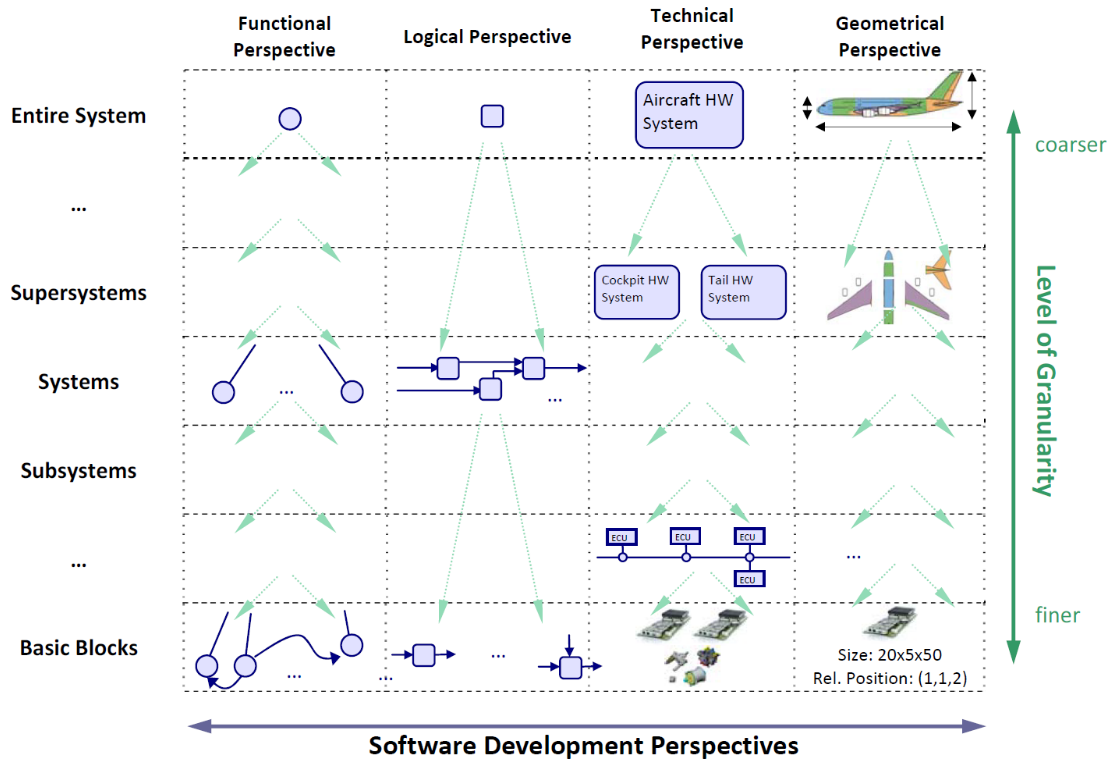

Figure 3: The two abstraction dimensions of the SPES Metamodel (Source [14, 26])

Platform services

A platform is much more than just a vehicle of information. It probably has the job of managing huge amounts of information, without neglecting their integrity and confidentiality. It moreover has to mediate between systems of inherently diverse nature. Worth considering for the realisation of these services is the solution proposed by the middleware Chromosome; see [6] and also [17]. Chromosome returns the control over the functionality of an application to the developer, by “hiding” the complexity and ensuring extensibility by plug & play mechanisms also at runtime. Chromosome moreover puts real-time capabilities at disposal.

4.3 Cooperation Design

Cooperation between single systems and CPS (i.e., cooperation at any level, see Sect. 3) can be organised considering the concepts of navigation space and navigational structure; see [16]. This means, navigation nodes and navigability between nodes are notions orthogonal to component and communication between components. Navigation moreover can (but not necessarily does) carry information. An intuitive example is given by smartphones, where for instance email reading can be interrupted by an incoming call: once the call ended, the control automatically returns to the previous screen and the user can continue reading his/her email. Information navigates when, e.g., within the email an address is selected and the smartphone offers to look up the address using a map service, to search for a connection with public transportation, etc.

Service delegation can moreover occur dynamically. This means, the situation can be assessed and required services be searched for in the surroundings of the system. With regards to the concepts of navigation nodes and navigability, the structure can spontaneously be redefined and/or rearranged.

Typically, a navigation structure is based on the services structure. The former, however, can omit some services, i.e., not necessarily all services are reflected –or represented– as a navigation node; an example hereof might be an antivirus. Navigation can also foresee shortcuts avoiding intermediate steps when these are, e.g., previously and univocally determined.

4.4 HCI Design

The success of web services lets us presume that, for the interaction of CPS with end users regardless of their education, age, gender, etc., the strategies used in web design can be reused. For this purpose, the models for presentation of [16] may be a good starting point; see also [12].

The presentation model is based on the navigational model, but addresses other challenges. Consider haptic in case of an amputee, or instructions imparted to hearing impaired, etc. These considerations greatly impact on the acceptability of CPS; see [4, 13].

5 Conclusions and outlook

The design-operation life-cycle continuum of CPSE reminds of a family album, where snapshots are memorised but in fact the portrayed subjects might exist beyond the ends of the album, i.e., some exist before the first (in chronological order) photograph was taken, some exist further after the date of the last photograph, and some others happen to appear as grown-ups when, e.g., a family member marries. Moreover, between two chronologically subsequent photographs, any family member has undergone a number of more or less slight changes.

Referring back to the SPES Metamodel, the first to catch one's eye is the compartmental division between software development perspectives: although the time unfolds from left to right, it is not to be understood that all levels of granularity of a CPS evolve simultaneously from one perspective to the next one. The picture misses moreover the correlation between the components across the different perspectives. On these realisations and considering the discussion above we plan to work out a process and a metamodel for CPS and to iteratively validate them by means of case studies.

In the framework of the programme "Research for Tomorrow's Production", sponsored by the German Federal Ministry of Education and Research, a project will be carried out whose research focus is the operation of CPS for the maintenance in a real production environment. The project S-CPS aims to bring together the data sensored at the equipment and installations of a production system that concern their diagnosis and the pertinent maintenance data, in order to automatically and dynamically generate out of them an overview of the necessary and free resources and the required skills of the staff to be involved as well as any further essential information, and present this overview in a so-called *resources' cockpit*; see Fig. 4. The resulting CPS will be instantiated in three very different contexts, namely for an automobile manufacturer, a car parts supplier, and a wind turbine manufacturer. This first case study will certainly provide valuable insights into the specific challenges associated with the engineering of complex, adaptive, distributed CPS, and the means to master them.

References

[1] Wil M. P. Aalst, Arthur H. M. Hofstede, and Mathias Weske. Business Process Management: A Survey. In Wil M. P. Aalst and Mathias Weske, editors, *Business Process Management International Conference (BPM'03, Proceedings)*, volume 2678 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2003.

Figure 4: Vision of the *resources' cockpit* in the Experimental and Digital Factory of the Technical University Chemnitz (Photo: Institut für Betriebswissenschaften und Fabriksysteme, TU Chemnitz)

[2] Esteve Almirall and Jonathan Wareham. Living Labs and open innovation: Roles and applicability. *The Electronic Journal for Virtual Organizations and Networks*, 10(3):21–46, 2008. Special Issue on Living Labs.

[3] Albert Angehrn. Designing mature internet business strategies: The ICDT model. *European Management Journal*, 15(4):361–369, August 1997.

[4] Manfred Broy, Eva Geisberger, María Victoria Cengarle, Patrick Keil, Jürgen Niehaus, Christian Thiel, and Hans-Jürgen Thönnißen-Fries. *Cyber-Physical Systems: Innovationsmotor für Mobilität, Gesundheit, Energie und Produktion*. Number 8 in acatech BEZIEHT POSITION. Springer, Berlin, 2012.

[5] Christian Buckl, Harald Rueß, and Bernhard Schätz. Cyber-Physical Systems: From Building to Evolving — A Radically New Engineering Challenge of European Dimension. Presentation at the EIT ICT KIC "Cyber-Physical Systems", February 2012.

[6] CHROMOSOME in 120 Minutes. Technical report, fortiss GmbH, April 2012.

[7] Paul Clements and Linda Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley Professional, 3rd edition, 2001.

[8] Ivica Crnkovic. Component-based approach for Embedded Systems. In *9th International Workshop on Component-Oriented Programming (WCOP'04, Proceedings)*, 2004.

[9] Bill Curtis, Marc Kellner, and Jim Over. Process Modeling. *Commununications of the ACM*, 35(9):75–90, 1992.

[10] Patricia Derler, Edward Lee, and Alberto Sangiovanni Vincentelli. Modeling Cyber-Physical Systems. *Proceedings of the IEEE*, 100(1):13–28, January 2012.

[11] Benoît Dutilleul, Frans Birrer, and Wouter Mensink. Unpacking European Living Labs: Analysing Innovation's Social Dimensions. *Central European Journal of Public Policy*, 4(1):60–85, 2010.

[12] Franca Garzotto and Vito Perrone. On the Acceptability of Conceptual Design Models for Web Applications. In Manfred Jeusfeld and Oscar Pastor, editors, *Conceptual Modeling for Novel Application Domains (ER'03 Workshops ECOMO, IWCMQ, AOIS, and XSDM, Proceedings)*, volume 2814 of *Lecture Notes in Computer Science*, pages 92–104. Springer, 2003.

[13] Eva Geisberger, Manfred Broy, María Victoria Cengarle, Patrick Keil, Jürgen Niehaus, Christian Thiel, and Hans-Jürgen Thönnißen-Fries. *agendaCPS: Integrierte Forschungsagenda Cyber-Physical Systems*. Springer, Berlin, 2012.

[14] Alexander Harhurin, Florian Hölzl, and Thomas Kofler. SPES Metamodel. Deliverable D1.2.B-6, Software Plattform Embedded Systems (SPES) 2020, December 2010.

[15] Drew Hemment, Rebecca Ellis, and Brian Wynne. Participatory Mass Observation and Citizen Science. *Leonardo*, 44(1):61–63, 2011.

[16] Rolf Hennicker and Nora Koch. A UML-based Methodology for Hypermedia Design. In Andy Evans, Stuart Kent, and Bran Selic, editors, *UML 2000 :The Unified Modeling Language, Advancing the Standard (3rd International Conference, Proceedings)*, volume 1939 of *Lecture Notes in Computer Science*, pages 410–424. Springer, 2000.

[17] Kai Huang, Gang Chen, Nadine Keddis, Michael Geisinger, and Christian Buckl. Demo Abstract: An Inverted Pendulum Demonstrator for Timed Model-Based Design of Embedded Systems. In *Third International Conference on Cyber-Physical Systems (ICCPS'12, Proceedings)*, page 224. IEEE Computer Society, 2012.

[18] Gabor Karsai, Janos Sztipanovits, Ákos Lédeczi, and Ted Bapty. Model-Integrated Development of Embedded Software. *Proceedings of the IEEE*, 91(1):145–164, January 2003.

[19] Mary Meldrum and Jeremy Rose. Activity Based generation of requirements for web-based information systems: the SSM/ICDT approach. In Timo Leino, Timo Saarinen, and Stefan Klein, editors, *The European IS Profession in the Global Networking Environment, 13th European Conference on Information Systems (ECIS'04, Proceedings)*, pages 1212–1223, 2004.

[20] Daniel Méndez Fernández, Klaus Lochmann, Birgit Penzenstadler, and Stefan Wagner. A case study on the application of an artefact-based requirements engineering approach. In *Evaluation Assessment in Software Engineering (EASE'11, Proceedings)*, pages 104–113. IEEE Computer Society, 2011.

[21] Milena Reichel and Heidi Schelhowe. Living labs: driving innovation through civic involvement. In Justine Cassell, editor, *7th International Conference on Interaction Design and Children (IDC'08, Proceedings)*, pages 141–144. ACM, 2008.

[22] Jeremy Rose. Soft systems methodology as a social science research tool. *Systems Research and Behavioral Science*, 14(4):249–258, July/August 1997.

[23] Tom Sorel. The Life Cycle Continuum. *Public Roads*, 68(1), July/August 2004.

[24] Janos Sztipanovits. Model Integration and Cyber-Physical Systems: A Semantics Perspective. Invited talk at FM'2011, June 2011. Joint work with Ted Bapty and Gabor Karsai and Sandeep Neema.

[25] Victor Teglasi. Why Transportation Mega-Projects (Often) Fail? Case Studies of Selected Transportation Mega-Projects in the New York City Metropolitan Area. Master's thesis, Columbia University, New York, USA, 2012.

[26] Judith Thyssen, Daniel Ratiu, Wolfgang Schwitzer, Alexander Harhurin, Martin Feilkas, and Eike Thaden. A System for Seamless Abstraction Layers for Model-based Development of Embedded Software. In Gregor Engels, Markus Luckey, Alexander Pretschner, and Ralf Reussner, editors, *Software Engineering Workshops 2010 (Proceedings)*, volume 160 of *Lecture Notes in Informatics*, pages 137–148. Gesellschaft für Informatik, 2010.

[27] Andrés Vignaga, Frédéric Jouault, María Cecilia Bastarrica, and Hugo Brunelière. Typing artifacts in megamodeling. *Software & Systems Modeling*, 12(1):105–119, 2013.

[28] Yingxu Wang and Antony Bryant. Process-Based Software Engineering: Building the Infrastructures. *Annals of Software Engineering*, 14(1-4):9–37, 2002.

[29] Martin Wirsing and Matthias Hölzl, editors. *Rigorous Software Engineering for Service-Oriented Systems*, volume 6582 of *Lectuer Notes in Computer Science*. Springer, 2011.

Author Index

Asplund, Fredrik
Axelsson, Jakob

Becker, Klaus
Behere, Sagar
Biehl, Matthias

Calinescu, Radu
Cengarle, María Victoria
Chen, Dejiu

El-Khoury, Jad

Fontanelli, Daniele

Gerasimou, Simos
Gonçalves Vieira Ferreira, Mauricio

Johnson, Kenneth

Kobetski, Avenir

Loiret, Frederic

Moro, Federico

Palopoli, Luigi

Rafiq, Yasmin
Rizano, Tizar
Romero, Alessandro
Roscoe, Bill

Schneider, Klaus

Törngren, Martin

Voss, Sebastian