# Self-Adaptation in Software-Intensive Cyber-Physical Systems: from System Goals to Architecture Configurations

Ilias Gerostathopoulos[1]
gerostat@in.tum.de

Tomas Bures[2,3]
bures@d3s.mff.cuni.cz

Petr Hnetynka[2]
hnetynka@d3s.mff.cuni.cz

Jaroslav Keznikl[2,3]
keznikl@d3s.mff.cuni.cz

Michal Kit[2]
kit@d3s.mff.cuni.cz

Frantisek Plasil[2]
plasil@d3s.mff.cuni.cz

Noël Plouzeau[4]
noel.plouzeau@irisa.fr

[1]Technische Universität München
Fakultät für Informatik
Munich, Germany

[2]Charles University in Prague
Faculty of Mathematics and Physics
Prague, Czech Republic

[3]Insitute of Computer Science
Academy of Sciences of the Czech Republic
Prague, Czech Republic

[4]IRISA
University of Rennes 1
Rennes, France

## ABSTRACT

Design of self-adaptive software-intensive Cyber-Physical Systems (siCPS) operating in dynamic environments is a significant challenge when a sufficient level of dependability is required. This stems partly from the fact that the concerns of self-adaptivity and dependability are to an extent contradictory. In this paper, we introduce IRM-SA (Invariant Refinement Method for Self-Adaptation) – a design method and associated formally grounded model targeting siCPS – that addresses self-adaptivity and supports dependability by providing traceability between system requirements, distinct situations in the environment, and predefined configurations of system architecture. Additionally, IRM-SA allows for architecture self-adaptation at runtime and integrates the mechanism of predictive monitoring that deals with operational uncertainty. As a proof of concept, it was implemented in DEECo, a component framework that is based on dynamic ensembles of components. Furthermore, its feasibility was evaluated in experimental settings assuming decentralized system operation.

## Keywords

Cyber-physical systems; Self-adaptivity; Dependability; System design; Component architectures

## 1. INTRODUCTION

*Cyber-physical systems* (CPS) are characterized by a network of distributed interacting elements which respond, by sensing and actuating, to activities in the physical world (their environments). Compared to traditional embedded systems, CPS are becoming more modular, dynamic, networked, and large-scale. For already a long time, many CPS have been also increasingly dependent on software, their most intricate and extensive constituent [40] – so that it is natural to talk about *software-intensive* CSP (siCPS). In this paper, we consider a class of siCPS that are distributed at a large scale and inherently dynamic. Examples of siCPS are numerous: systems for intelligent car navigation, smart electric grids, emergency coordination systems, to name just a few.

Designing such systems is a challenging task, as one has to deal with the different, and to an extent contradictory, concerns of dependability and self-adaptivity. Since they often host safety-critical applications, they need to be dependable (safe and predictable in the first place), even when being highly dynamic. Since siCPS operate in diverse environments (parts of the ever-changing physical world), they need be self-adaptive [54]. An additional issue is the inherent operational uncertainty related to their infrastructure. Indeed, siCPS need to remain operable even in adversarial conditions, such as network unavailability, hardware failures, resource scarcity, etc.

Achieving synergy of dependability and self-adaptivity in presence of operational uncertainty is hard. Existing approaches typically succeed in addressing just one aspect of the problem. For example, agent-oriented methodologies address conceptual autonomy [22, 56]; component-based mode-switching methods bring dependability assurances with limited self-adaptivity [29, 39]. Operational uncertainty is typically viewed as a separate problem [19]. What is missing is a design method and associated model that would specifically target the development of dependable and self-adaptive siCPS while addressing operational uncertainty.

Self-adaptive siCPS need to be able to adapt to distinct runtime *situations* in the environment (i.e., its states as observed by the system). This takes the form of switching between distinct architecture *configurations* of the system (*products* in SPLs [3]). Being reflected in system requirements, these configurations and the associated situations can be systematically identified and addressed via requirement analysis and elaboration, similar to identification of adaptation scenarios in a Dynamically Adaptive System (DAS) [32].

A problem is that an exhaustive enumeration of configurations and situations at design time is not a viable solution in the domain of siCPS, where unanticipated situations can appear in the environment (*external uncertainty* in [24]). Moreover, another challenge is that self-adaptive siCPS need to base their adaptation actions not only on the current situation, but also on how well the system can currently operate, e.g., whether certain components can properly communicate (issue similar to agent capabilities [72]).

In this paper we tackle these challenges by proposing an extension to IRM [47] – a design method and associated formally grounded model targeting siCPS requirements and architecture design. This extension (*IRM for Self-Adaptation – IRM-SA*) supports self-adaptivity and, at the same time, accounts for dependability. In particular, dependability is supported primarily at design time

through traceability between system requirements and configurations. Self-adaptivity is supported in the form of defining alternative configurations at design time and switching between them at runtime (*architecture self-adaptation*) to address specific situations. In order to select a configuration at runtime, SAT solving is employed.

In general, siCPS are networked control systems facing difficult open issues both in terms of control theory [37] and decentralized decision-making [8]. In this paper, we focus on a large class of siCPS for which the pace of changes and temporary disagreements among nodes do not prevent a fully decentralized self-adaptation. In Section 7 we discuss these issues in more detail and describe criteria under which a siCPS is amenable to our approach.

To evaluate the feasibility of IRM-SA we have applied it on a firefighter coordination case study – *Firefighter Tactical Decision System* (FTDS) – developed within the project DAUM[1]. As proof of the concept, we implemented self-adaptation based on IRM-SA by extending DEECo [15] – a component model facilitating open-ended and highly dynamic siCPS architectures. We also evaluated the design process of IRM-SA via a controlled experiment.

## 1.1 Contributions

In summary, key contributions of this paper include:

i. The description of a design method and associated model that allow modeling design alternatives in the architecture of a siCPS pertaining to distinct situations via systematic elaboration of system requirements;

ii. An automated self-adaptation method that selects the appropriate architecture configuration based on the modeled design alternatives and the perceived situation;

iii. An evaluation of how well the proposed self-adaptation method deals with operational uncertainty via predictive monitoring;

iv. A discussion of strategies to deal with unanticipated situations at design time and at runtime.

## 1.2 Structure

The paper is structured as follows. Section 2 describes the running example, while Section 3 presents the background on which IRM-SA is based. Then, Section 4 overviews the core ideas of IRM-SA. Section 5 elaborates on the modeling of different design alternatives in IRM-SA by extending IRM, while Section 6 focuses on the selection of applicable architecture configurations at runtime. Section 7 focuses on the intricacies of self-adaptation in distributed settings and describes criteria under which self-adaptation can be performed in a decentralized manner, together with an example case. Section 8 details on our prototype implementation of IRM-SA in DEECo. Section 9 reports on an empirical study of IRM-SA effectiveness via a controlled experiment. Section 10 discusses the mechanisms to cope with unanticipated situations and the generality of IRM-SA. Section 11 positions our work with respect to the state of the art, while Section 12 concludes the paper and outlines some yet-to-be-addressed challenges.

[1] http://daum.gforge.inria.fr/

# 2. RUNNING EXAMPLE

In this paper, we use as running example a simple scenario from the FTDS case study, which was developed in cooperation with professional firefighters. In the scenario, the firefighters belonging to a tactical group communicate with their group leader. The leader aggregates the information about each group member's condition and his/her environment (parameters considered are firefighter *acceleration*, external *temperature*, *position* and *oxygen level*). This is done with the intention that the leader can infer whether any group member is in danger so that specific actions are to be taken to avoid casualties.

On the technical side, firefighters in the field communicate via low-power nodes integrated into their personal protective equipment. Each of these nodes is configured at runtime depending on the task assigned to its bearer. For example, a hazardous situation might need closer monitoring of a certain parameter (e.g., temperature). The group leaders are equipped with tablets; the software running on these tablets provides a model of the current situation (e.g., on a map) based on the data aggregated from the low-power nodes.

The main challenge of the case study is how to ensure that individual firefighters (nodes) retain their (a) autonomy so that they can operate in any situation, even entirely detached from the network and (b) autonomicity so that they can operate optimally without supervision, while still satisfying certain system-level constraints and goals. Examples of challenging scenarios include (i) loss of communication between a leader and members due to location constraints, (ii) malfunctioning of sensors due to extreme conditions or battery drainage, and (iii) data inaccuracy and obsoleteness due to intermittent connections. In all these cases, firefighters have to adjust their behavior according to the latest information available. Such adjustments range from simple adaptation actions (e.g., increasing the sensing rate in face of a danger) to complex cooperative actions (e.g., relying on the nearby nodes for strategic actions when communication with the group leader is lost).

# 3. BACKGROUND

*Invariant Refinement Method* (IRM) [47] is a goal-oriented design method targeting the domain of siCPS. IRM builds on the idea of iterative refinement of system objectives yielding low-level obligations which can be operationalized by system components. Contrary to common goal-oriented modeling approaches (e.g., KAOS [50], Tropos/i* [12]), which focus entirely on the problem space and on stakeholders' intentions, IRM focuses on system components and their contribution and coordination in achieving system-level objectives. IRM also incorporates the notion of feedback loops present in autonomic and control systems, i.e., all "goals" in IRM are to be constantly maintained, not achieved just once. A key advantage of IRM is that it allows capturing the compliance of design decisions with the overall system goals and requirements; this allows for design validation and verification.

The main idea of IRM is to capture high-level system goals and requirements in terms of *invariants* and, by their systematic refinement, to identify system *components* and their desired interaction. In principle, invariants describe the *operational normalcy* of the system-to-be, i.e., the desired state of the system-to-be at every time instant. For example, the main goal of our running example is expressed by INV-1: "GL keeps track of the condition of his/her group's members" (Figure 1).

IRM invariants are agnostic on the language used for their specification. In this paper, plain English is used for simplicity's

sake; passive voice has been chosen in order to adopt a more descriptive than prescriptive style. Other possible choices include adopting a style based on SHALL statements commonly used in requirements specifications, or a complete textual requirements specification language, such as RELAX [77].

In general, invariants are to be maintained by system components and their cooperation. At the design stage, a component is a participant/actor of the system-to-be, comprising internal state. Contrary to common goal-oriented approaches (e.g., [51], [12]), only software-controlled actors are considered. The two components identified in the running example are Firefighter and Officer.

As a special type of invariant, an *assumption* describes a condition expected to hold about the environment; an assumption is not expected to be maintained by the system-to-be. In the example, INV-8 in Figure 1 expresses what the designer assumes about the monitoring equipment (e.g., GPS).

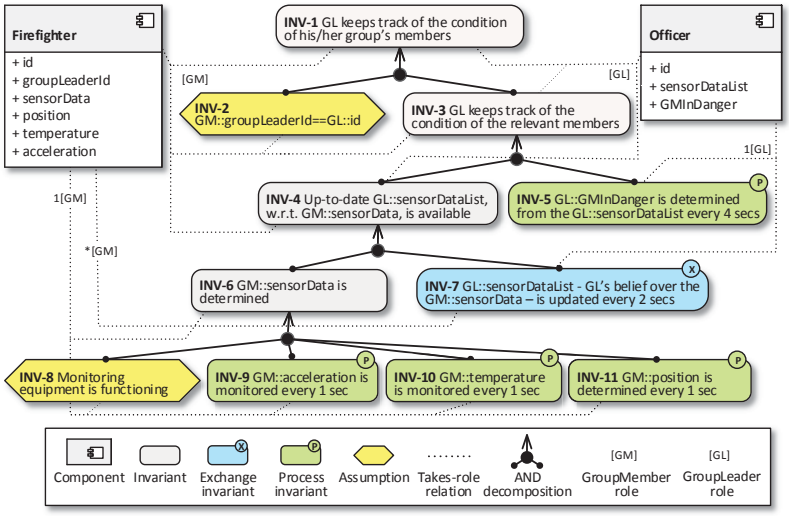


**Figure 1: IRM decomposition of the running example.**

As a design decision, the identified top-level invariants are decomposed via so-called *AND-decomposition* into conjunctions of more concrete sub-invariants represented by a decomposition model – *IRM model*. Formally, the IRM model is a directed acyclic graph (DAG) with potentially multiple top-level invariants, expressing concerns that are orthogonal. The *AND-decomposition* is essentially a refinement, where the composition (i.e., conjunction) of the children implies the fact expressed by the parent (i.e., the fact expressed by the composition is in general a specialization, following the traditional interpretation of refinement). Formally, an AND-decomposition of a parent invariant $I_p$ into the sub-invariants $I_{s1}, \ldots, I_{sn}$ is a refinement, if it holds that:

1. $I_{s1} \wedge \ldots \wedge I_{sn} \Rightarrow I_p$ (entailment)
2. $I_{s1} \wedge \ldots \wedge I_{sn} \not\Rightarrow false$ (consistency)

For example, the top-level invariant in Figure 1 is refined to express the necessity to keep the list of sensor data updated on the Officer's side (INV-4) and the necessity to filter the data to identify group members that are in danger (INV-5).

Decomposition steps ultimately lead to a level of abstraction where leaf invariants represent detailed design of the system constituents. There are two types of leaf invariants: *process invariants* (labeled P, e.g., INV-5) and *exchange invariants* (labeled X, e.g., INV-7). A process invariant is to be maintained by a single component (at runtime, in particular by a cyclic process manipulating the component's state – Section 8.1). Conversely, exchange invariants are maintained by component interaction, typically taking the form of knowledge exchange within a group of components (Section 8.1). In this case, exchange invariants express the necessity to keep a component's belief over another component's internal state. Here, *belief* is defined as a snapshot of another component's internal state [47] – often the case in systems of autonomous agents [72]; inherently, a belief can get outdated and needs to be systematically updated by knowledge exchange in the timeframe decided at the design stage.

# 4. IRM-SA – THE BIG PICTURE

To induce self-adaptivity by design so that the running system can adapt to situations, it is necessary to capture and exploit the architecture variability in situations that warrant self-adaptation. Specifically, the idea is to identify and map *applicable configurations* to situations by elaborating *design alternatives* (alternative realizations of system's requirements); then these applicable configurations can be employed for architecture adaptation at runtime.

Therefore, we extended the IRM design model and process to capture the design alternatives and applicable configurations along with their corresponding situations. For each situation there can be one or more applicable configurations. To deal with operational uncertainty, we also extend the model by reasoning on the inaccuracies of the belief.

At runtime, the actual architecture self-adaptation is performed via three recurrent steps: (i) determining the current situation, (ii) selecting one of the applicable configurations, and (iii) reconfiguring the architecture towards the selected configuration.

The challenge is that mapping configurations to situations typically involves elaborating a large number of design alternatives. This creates a scalability issue both at design-time and runtime, especially when the individual design alternatives have mutual dependencies or refer to different and possibly nested levels of abstraction. To address scalability, we employ (i) separation of concerns via decomposition at design time; (ii) a formal base of the IRM-SA design model and efficient reasoning based on SAT solving for the selection of applicable configurations at runtime (Section 6).

# 5. MODELING DESIGN ALTERNATIVES

## 5.1 Concepts addressing Self-Adaptivity

Although providing a suitable formal base for architecture design via elaboration of requirements, IRM in its pure form does not allow modeling of design alternatives.

Therefore, we extended the IRM design model with the concepts of alternative decomposition – *OR-decomposition* – and *situation*. The running example as modelled in IRM-SA is depicted in Figure 2.

Essentially, OR-decomposition denotes a variation point where each of the children represents a design alternative. Technically, OR-decomposition is a refinement, where each of the children individually implies (i.e., refines) the fact expressed by the parent. OR-decompositions can be nested, i.e., a design alternative can be further refined via another OR-decomposition. Formally, an OR-

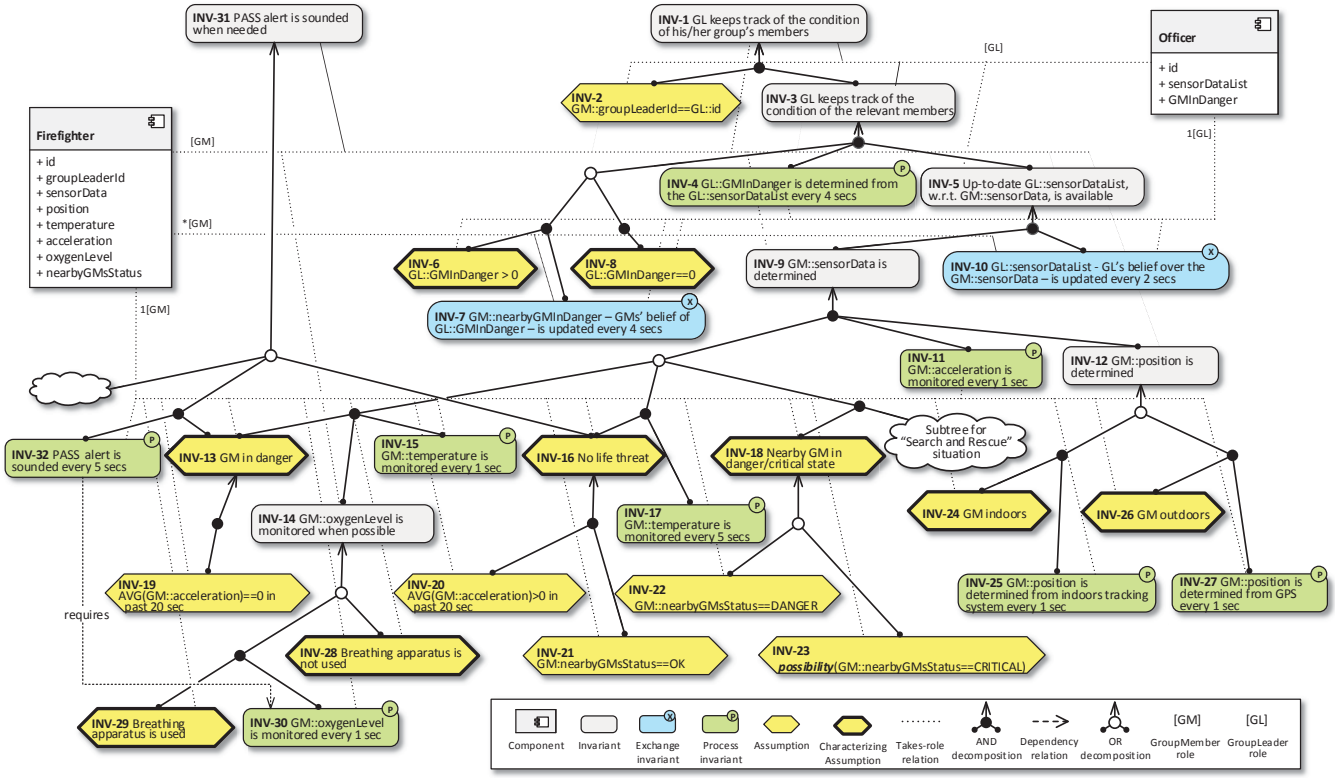


**Figure 2: IRM-SA model of the running example.**

decomposition of a parent invariant $I_p$ into the sub-invariants $I_{s1}, \ldots, I_{sn}$ is a refinement if it holds that:

1. $I_{s1} \vee \ldots \vee I_{sn} \Rightarrow I_p$ (alternative entailment)
2. $I_{s1} \vee \ldots \vee I_{sn} \not\Rightarrow false$ (alternative consistency)

Each design alternative addresses a specific situation which is characterized via an assumption. Thus, each $I_{si}$ contains at its top-most level a *characterizing assumption* as illustrated in Figure 2.

For example, consider the left-most part of the refinement of INV-3: "GL keeps track of the condition of the relevant members", which captures two design alternatives corresponding to the situations where either some firefighter in the group is in danger or none is. In the former case (left alternative), INV-7 is also included – expressing the necessity to inform the other firefighters in the group that a member is in danger. In this case, INV-6 and INV-8 are the characterizing assumptions in this OR-decomposition.

The situations addressed in an OR-decomposition may overlap, i.e., their charactering assumptions can hold at the same time. This is the case of INV-13 and INV-18 capturing that both one Firefighter is in danger and a nearby colleague as well. Consequently, there are more than one applicable configurations and therefore a prioritization is needed (Section 6.2). As an aside, allowing situations in an OR-decomposition to overlap also provides a built-in fault-tolerance mechanism (Section 10.1.1).

Technically, if a design alternative in an OR-decomposition is further refined in terms of an AND-decomposition (or vice-versa), we omit the invariant representing the alternative and connect the AND-decomposition directly to the OR-decomposition to improve readability (e.g., design alternatives of INV-12).

We distinguish two kinds of invariants: *computable* and *non-computable*. While a computable invariant can be programmatically evaluated at runtime, a non-computable invariant serves primarily for design and review-based validation purposes. Thus, the characterizing assumptions need to be either computable or decomposed into computable assumptions.

An example of a non-computable characterizing assumption is INV-16: "No life threat". It is AND-decomposed into the computable assumptions INV-20 and INV-21, representing two orthogonal concerns, which can be evaluated by monitoring the Firefighter's internal state.

Dependencies may also exist between invariants in design alternatives across different OR-decompositions (*cross-tree dependencies*), reflecting constraints of the physical world. These dependencies are captured in the IRM-SA model by directed links between invariants labeled with "requires", resp. "collides", which capture the constraint that the source invariant can appear in a configuration only with, resp. without, the target invariant. For example, in order for INV-32 to appear in a configuration, INV-30 has to be included as well, capturing the real-life constraint where the *Personal Alert Safety System* (PASS) is attached to the *self-contained breathing apparatus* (SCBA) of a firefighter; thus if the SCBA is not used, then the PASS cannot be used as well. The "collides" dependency is not illustrated in our running example.

## 5.2 Concepts Addressing Dependability

In IRM-SA, dependability is mainly pursued by tracing the low-level processes to high-level invariants. Moreover, to deal with the operational uncertainty in dynamic siCPS, IRM-SA goes beyond the classical goal-modeling approaches and allows self-adaptation based not only on valuations of belief (snapshot of a remote component's internal data), but also on valuations of

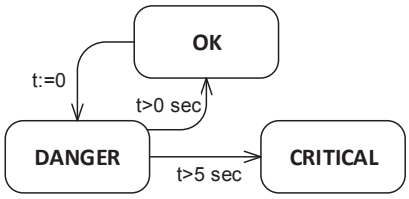


**Figure 3: Timed automaton capturing the transitions in the valuation of the nearbyGMsStatus field.**

associated metadata (timestamp of belief, timestamp of sending/receiving the belief over the network, etc.). This functionality also adds to the dependability by self-adapting in anticipation of critical situations. Nevertheless, IRM-SA support for dependability does not cover other dependability aspects, such as privacy and security.

A key property here is that a belief is necessarily outdated, because of the distribution and periodic nature of real-time sensing in siCPS. For example, the position of a Firefighter as perceived by his/her Officer would necessarily deviate from the actual position of the Firefighter if he/she were on the move. Instead of reasoning directly on the degree of belief outdatedness (on the time domain), we rely on models that predict the evolution of the real state (e.g., state-space models if this evolution is governed by a physical process), translate the outdatedness from the time domain to the data domain of the belief (e.g., position inaccuracy in meters) and reason on the degree of *belief inaccuracy*. We call this type of reasoning *proactive reasoning*. To enable proactive reasoning, we build on our previous work in quantifying the degree of belief inaccuracy in dynamic siCPS architectures [1].

For illustration, consider an assumption "*inaccuracy*(GM::position) < 20 m", which models the situation where the difference of the measured and actual positions is less than 20 meters. In this case, belief inaccuracy is both (i) inherent to the sensing method (more GPS satellites visible determine more accurate position), and (ii) related to the network latencies when disseminating the position data (more outdated data yield more inaccurate positions – since firefighters are typically on the move, their position data are subject to outdating). As a result, an Officer has to reason on the cumulative inaccuracy of the position of his/her Firefighter.

When the domain of the belief field is discrete instead of continuous, we rely on models that capture the evolution of discrete values in time, such as timed automata. For illustration, consider assumption INV-23: "*possibility*(GM::nearbyGMsStatus == CRITICAL)", which models the situation where the nearbyGMsStatus enumeration field with values OK, DANGER, and CRITICAL is possible to evaluate to CRITICAL. This presumes that the designer relies on a simple timed automaton such as the one depicted in Figure 3, which encodes the domain knowledge that a firefighter gets into a critical situation (and needs rescuing) at least 5 seconds after he/she gets in danger.

All in all, the invariants that are formulated with *inaccuracy* and *possibility* provide a fail-safe mechanism for adversarial situations, when the belief of a component gets so inaccurate that special adaptation actions have to be triggered. This proactive reasoning adds to the overall dependability of the self-adaptive siCPS.

## 5.3 The Modeling Process

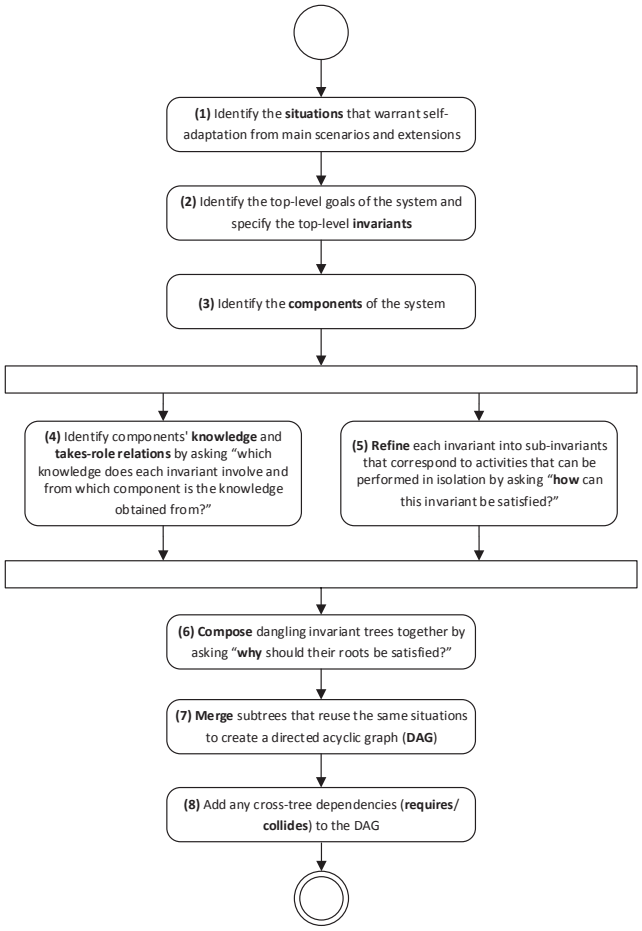


**Figure 4: Steps in the IRM-SA modeling process.**

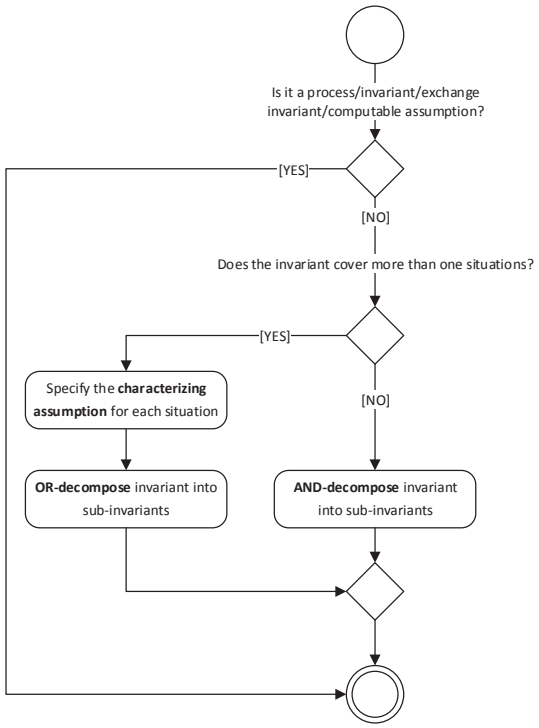


**Figure 5: Steps in a single invariant refinement.**

As is usually the case with software engineering processes, IRM-SA modeling process is a mixed top-down and bottom-up process. As input, the process requires a set of use cases/user stories covering both the main success scenarios and the associated extensions. The main steps of the process are illustrated in Figure 4. After the identification of the main situations, goals and components, the architect starts to specify the knowledge of each component together with its takes-role relations (step 4), while, in parallel, he/she starts refining the invariants (step 5). These two steps require potentially several iterations to complete. In step 6, the architect composes the dangling invariant trees that may have resulted from the previous steps, i.e., the trees the roots of which are *not* top-level invariants. Contrary to the previous steps, this is a bottom-up design activity. In the final steps, as an optimization, the subtrees produced in the previous steps that are identical are merged together, and "requires"/"collides" dependencies are added. The result is a DAG – this optimization was applied also in Figure 2.

The workings of a single refinement are depicted in Figure 5. Based on whether the invariant under question is to be satisfied in a different way in different situations (e.g., "position reading" will be satisfied by using the GPS sensor when outdoors and by using the indoor positioning system when indoors), the architect chooses to refine the invariant by OR- or AND-decomposition. Obviously, in the former case, the refinement involves specifying the characterizing assumption for each design alternative (situation). Note that, if the characterizing assumption is not computable, it will get refined in a next step as any other invariant in such a case.

The process of systematic identification of all the possible variation points and modeling the corresponding design alternatives and situations is closely related to the identification of adaptation scenarios in a Dynamically Adaptive System (DAS) [32]. Here, one can leverage existing approaches in requirements engineering ranging from documentation of main use-case scenarios and extensions to obstacle/threat analysis on goal models [52]. Performance and resource optimization concerns can also guide the identification of variation points and corresponding design alternatives.

For example, the rationale behind the OR-decomposition of the left-most part of the AND-decomposition of INV-9 is resource optimization: under normal conditions the accuracy of external temperature monitoring can be traded off for battery consumption of the low-power node; this, however, does not hold in danger (e.g., a firefighter is not moving, INV-19), when higher accuracy of external temperature monitoring is needed.

On the contrary, the OR-decomposition of INV-12 has its rationale in a functional constraint: since GPS is usually not available within a building, a firefighter's position has to be monitored differently in such a case, e.g., through an indoors tracking system [17]. This is an example of a technology-driven process of identification of design alternatives, where the underlying infrastructure significantly influences the possible range of adaptation scenarios [32]. For example, it would not make sense to differentiate between the situations of being indoors and outdoors, if there were no way to mitigate the "GPS lost signal" problem using the available infrastructure.

This highlights an important point: IRM-SA allows for modeling the environment via assumptions, but, at the same time, guides the designer into specifying only the pertinent features of the environment, avoiding over-specification.

For a complete example of this modeling process, we refer the reader to the online IRM-SA User Guide[2]. To support the modeling process, we have also developed a prototype of a GMF-based IRM-SA design tool [42].

# 6. SELECTING ARCHITECTURE CONFIGURATIONS BY SAT SOLVING

As outlined in Section 4, given an IRM-SA model, the selection of a configuration for a situation can be advantageously done by directly reasoning on the IRM-SA model at runtime. In this section we describe how we encode the problem of selecting an applicable configuration into a Boolean satisfiability (SAT) problem (6.1), our prioritizing strategy with multiple applicable configurations (6.2), and how we bind variables in the SAT instance based on monitoring (6.3).

To simplify the explanation, we use the term "clause" in this section even for formulas which are not necessarily valid clauses in the sense of CNF (Conjunctive Normal Form – the default input format for SAT), but rely on the well-known fact that every propositional formula can be converted to an equisatisfiable CNF formula in polynomial time.

## 6.1 Applicable Configurations

Formally, the problem of selecting an applicable configuration is the problem of constructing a set $C$ of selected invariants from an IRM-SA model such that the following rules are satisfied: (i) all the top-level invariants are in $C$; (ii) if an invariant $I_p$ is decomposed by an AND-decomposition to $I_1, \dots, I_m$, then $I_p \in C$ iff all $I_1, \dots, I_m \in C$; (iii) if an invariant $I_p$ is decomposed by an OR-decomposition to $I_1, \dots, I_m$, then $I_p \in C$ iff at least one of $I_1, \dots, I_m$ is in $C$; (iv) if an invariant $I_p$ requires, resp. collides (with), $I_q$, then $I_p \in C$ iff $I_p \in C$, resp. $I_p \notin C$. The set C represents an applicable configuration. The rules above ensure that $C$ is well-formed with respect to decomposition and cross-tree dependencies semantics. Figure 6 shows a sample applicable configuration (selected invariants are outlined in grey background).

Technically, for the sake of encoding configuration selection as a SAT problem, we first transform the IRM-SA model to a forest by duplicating invariants on shared paths. (This is possible because the IRM-SA model is a DAG.) Then we encode the configuration

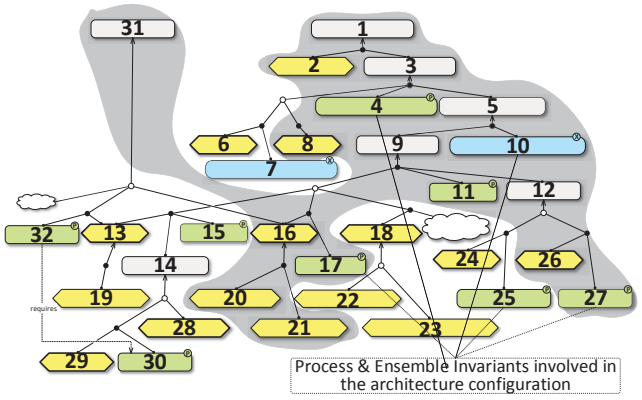


**Figure 6: An architecture configuration of the running example.**

[2] http://www.ascens-ist.eu/irm

$C$ we are looking for by introducing Boolean variables $s_1, \dots, s_n$, such that $s_i = true$ iff $I_i \in C$. To ensure $C$ is well-formed, we introduce clauses over $s_1, \dots, s_n$ reflecting the rules (i)-(iv) above. For instance, the IRM-SA model from Figure 2 will be encoded as shown in Figure 7, lines 1-26.

To ensure that $C$ is an applicable configuration w.r.t. a given situation, we introduce Boolean variables $a_1, \dots, a_n$ and add a clause $s_i \Rightarrow a_i$ for each $i \in \{1 \dots n\}$ (Figure 7, line 29). The value of $a_i$ captures whether the invariant $I_i$ is *acceptable*; i.e., $true$ indicates that it can be potentially included in $C$, $false$ indicates otherwise. The variables $a_1, \dots, a_n$ are bound to reflect the state of the system and environment (Figure 7, lines 31-39). This binding is described in Section 6.3.

In the resulting SAT instance, the variable $s_t$ for each top-level invariant $I_t$ is bound to true to enforce the selection of at least one applicable configuration. A satisfying valuation of such a SAT instance encodes one applicable configuration (or more than one in case of overlapping situations – see Section 6.2), while unsatisfiability of the instance indicates nonexistence of an applicable configuration in the current situation.

## 6.2 Prioritizing Applicable Configurations

Since the situations in an OR-decomposition do not need to be exclusive but can overlap, SAT solving could yield more than one applicable configurations. In this case, we assume a post-processing process that takes as input the IRM-SA model with the applicable configurations and outputs the selected configuration based on analysis of preferences between design alternatives. For this purpose, one can use strategies that range from simple total preorder of alternatives in each decomposition to well-established soft-goal-based techniques for reasoning on goal-models [20]. In the rest of the section, we detail on the prioritization strategy used in our experiments, which we view as just one of the many possible.

In our experiments (Section 7.2), we have used a simple prioritization strategy based on total preorder of design alternatives in each OR-decomposition. Here, for simplicity, a total preorder – numerical ranking – is considered (1 corresponds to the most preferred design alternative, 2 to the second most preferred, etc.). The main idea of the strategy is that the preferences loose significance by an order of magnitude from top to bottom, i.e., preferences of design alternatives that are lower in an IRM-SA tree cannot impact the selection of a design alternative that is above them on a path from the top-level invariant.

More precisely, given an IRM-SA tree, every sub-invariant $I_i$ of an OR-decomposition is associated with its OR-level number $d_i$, which expresses that $I_i$ is a design alternative of a $d_i$-th OR-decomposition on a path from the top-level invariant (level 1) to a leaf. For each OR-level, there is its cost base $b_j$ defined in the following way: (a) the lowest OR-level has cost base equal to 1, (b) the $j$-th OR-level has its cost base $b_j = b_{j+1} * (n_{j+1} + 1)$, where $n_{j+1}$ denotes the number of all design alternatives at the level $j+1$ (i.e., considering all OR-decomposition at this level). For example, the 2nd OR-level in the running example has $b_2 = b_3 * (n_3 + 1) = 1 * (4 + 1) = 5$, since the 3rd OR-level (lowest) has in total 4 design alternatives (2 from the OR-decomposition of INV-14 and 2 from that of INV-18).

Having calculated the base for each OR-level, the cost of a child invariant $I_i$ of a $d_i$-th OR-decomposition with a cost $b_{d_i}$ is defined as $rank * b_{d_i}$, where $rank$ denotes the rank of the design

1. // 1. configuration constraints based of the IRM model
2. // top level decomposition in Figure 2
3. $s_{11\_1} \wedge s_{27} \wedge s_{28} \Leftrightarrow s_{11}$ // $s_{11\_1}$ represents the anonymous invariant in the AND decomposition of INV-9
4. $s_{33\_1} \vee s_{33\_2} \vee s'_{20} \Leftrightarrow s_{33}$ // $s'_{20}$ is a copy of $s_{20}$
5. 
6. // decomposition level 1 in Figure 2
7. $s_{11\_1\_1} \vee s_{11\_1\_2} \vee s_{11\_1\_3} \Leftrightarrow s_{11\_1}$
8. $s_{28\_1} \vee s_{28\_2} \Leftrightarrow s_{28}$
9. $s_{34} \wedge s'_{13} \Leftrightarrow s_{33\_2}$ // $s'_{13}$ is a copy of $s_{13}$
10. 
11. // decomposition level 2 in Figure 2
12. $s_{13} \wedge s_{14} \wedge s_{15} \Leftrightarrow s_{11\_1\_1}$
13. $s_{20} \wedge s_{21} \Leftrightarrow s_{11\_1\_2}$
14. $s_{24} \wedge \dots \Leftrightarrow s_{11\_1\_3}$
15. $s_{30} \wedge s_{29} \Leftrightarrow s_{28\_1}$
16. $s_{32} \wedge s_{31} \Leftrightarrow s_{28\_2}$
17. 
18. // decomposition level 3 in Figure 2
19. $s_{16} \Leftrightarrow s_{13}$
20. $s'_{16} \Leftrightarrow s'_{13}$
21. $s_{14\_1} \vee s_{19} \Leftrightarrow s_{14}$
22. $s_{22} \wedge s_{23} \Leftrightarrow s_{20}$
23. … // similar for $s'_{20}, s_{24}$
24. 
25. // decomposition level 4 in Figure 2
26. $s_{17} \wedge s_{18} \Leftrightarrow s_{14\_1}$
27. 
28. // 2. only applicable invariants may be selected into a configuration
29. $(s_{11} \Rightarrow a_{11}) \wedge \dots \wedge (s_{32} \Rightarrow a_{32}) \wedge \dots$
30. 
31. // 3. determining acceptability according to monitoring
32. // (current configuration as shown in Figure 6)
33. // 3.1. active monitoring
34. $a_{11} = \cdots$ // true or false based on the monitoring of INV-9
35. … // repeat for $a_{16}, a'_{16}, a_{17}, a_{19}, a_{21}, a_{22}, a'_{22}, a_{23}, a'_{23}, a_{25}, a_{27}, a_{28}, a_{32}, a_{33}$
36. 
37. // 3.2. predictive monitoring
38. $a_{15} = \cdots$
39. … // repeat for the rest

**Figure 7: Encoding the IRM-SA model of running example into SAT.**

alternative that the invariant $I_i$ corresponds to. Finally, a simple graph traversal algorithm is the employed to calculate the cost of each applicable configuration as the sum of the cost of the selected invariants in the applicable configuration. The applicable configuration with the smallest cost is the preferred one – becomes the *current configuration*.

## 6.3 Determining Acceptability

Determining acceptability of an invariant $I_i$ (i.e., determining the valuation of $a_i$) is an essential step. In principle, a valuation of $a_i$ reflects whether $I_i$ is applicable w.r.t. the current state of the system and the current situation. Essentially, $a_i = false$ implies that $I_i$ cannot infer an applicable configuration.

We determine the valuation of $a_i$ in one of the following ways (alternatively):

(1) *Active monitoring*. If $I_i$ belongs to the current configuration and is computable, we determine $a_i$ by evaluating $I_i$ w.r.t. the current knowledge of the components taking a role in $I_i$.

(2) *Predictive monitoring*. If $I_i$ does not belong to the current configuration and is computable, it is assessed whether $I_i$ would be satisfied in another configuration if chosen.

In principle, if $I_i$ is not computable, its acceptability can be inferred from the computable sub-invariants.

For predictive monitoring, two evaluation approaches are employed: (a) The invariant to be monitored is extended by a

*monitor predicate* (which is translated into a monitor – Section 8.4) that assesses whether the invariant would be satisfied if selected, and (b) the history of the invariant evaluation is observed in order to prevent oscillations in current configuration settings by remembering that active monitoring found an invariant not acceptable in the past.

Certainly, (a) provides higher accuracy and thus is the preferred option. It is especially useful for process invariants, where the monitor predicate may assess not only the validity of process invariant (e.g., by looking at knowledge valuations of the component that take a role in it), but also whether the underlying process would be able to perform its computation at all. This can be illustrated on the process invariant INV-27, where the process maintaining it can successfully complete (and thus satisfy the invariant) only if GPS is operational and at least three GPS satellites are visible.

# 7. SOLVING THE SAT PROBLEM IN DISTRIBUTED SETTINGS

When implementing self-adaptation via IRM-SA in a distributed siCPS such as the firefighter tactical decision system, there are three main choices to consider:

1. ***Centralized self-adaptation (CA)***. A selected arbiter collects all the necessary knowledge from components, solves the SAT problem, and reliably communicates the result back to the components. This assumes that the system can be paused for the whole duration of the above process (strict synchronization), so that each component receives self-adaptation decisions that are relevant to its current state.
2. ***Decentralized self-adaptation with distributed consensus (DADC).*** Each component performs SAT solving locally based on its local knowledge (local view of system state), without requiring this knowledge to be synchronized across components. The results of SAT solving are then communicated and agreed upon between all components. This relaxes the assumption on strict synchronization, but still assumes that a component can be paused from the point that it solves the SAT problem until a consensus is built.
3. ***Decentralized self-adaptation with no distributed consensus (DANC).*** Similar to DADC, each component performs SAT solving locally based on its local knowledge, (local view of system state), without requiring this knowledge to be synchronized across nodes. However, the results of SAT solving are not communicated and no consensus is built. This allows components to keep their autonomy even detached from the network.

CA and DADC are both well-known solutions, documented in the state of the art of distributed systems [27, 28], multi-agent systems and autonomous agents [43, 62], and cooperative and networked control systems [33, 36]; they are also widely used in practice. They work well in systems with limited dynamicity, where network communication and the consequent timing issues (e.g., delays) can be mostly ignored or bounded. These assumptions are however not plausible in many siCPS, which are spatial systems deployed on top of ad-hoc wireless infrastructures with no communication guarantees and comprised of components that can dynamically appear and disappear. In such contexts, DANC is a more fitting choice, as it can be easily combined with proactive reasoning (Section 5.2) at the level of each individual component. This combination allows each component to make individual decisions that can help deal with threats related to network disconnections and delays.

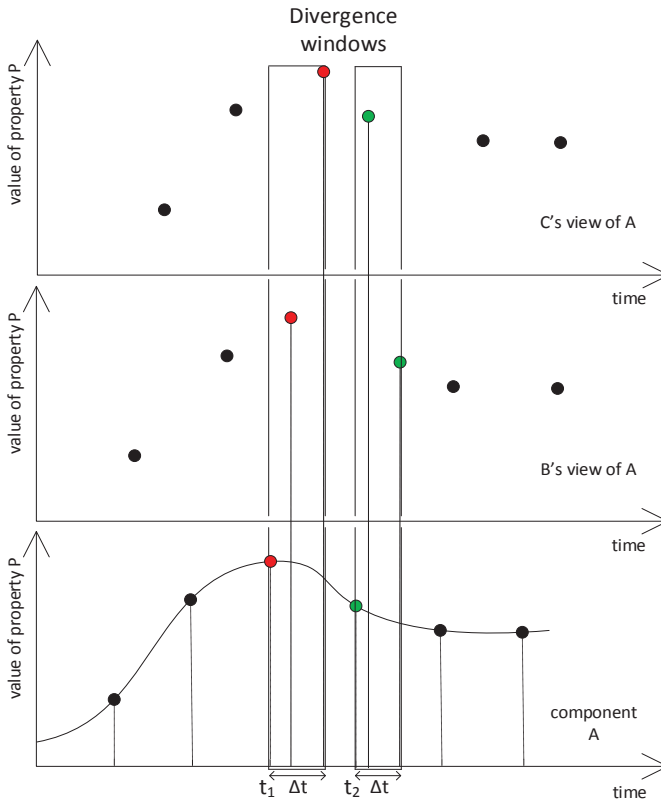


**Figure 8: Connection between outdated views and divergence windows.**

In the rest of this section we detail on DANC by identifying and formalizing the criteria that establish the perimeter of its applicability. We also exemplify the combination of DANC and proactive reasoning in our running example and discuss its ability to deal with threats related to network disconnections.

## 7.1 Decentralized Self-Adaptation with no Distributed Consensus

DANC can be employed in systems that are by nature resilient to situations when the system is de-synchronized. During de-synchronization, some nodes of the system arrive at different decisions due to their differently outdated knowledge (*divergence*). This can result into inconsistencies, e.g., cases where there are assumptions in the IRM-SA model that one component considers satisfied, while another considers violated. As an example, depicted in Figure 8, a property P of component A may reach some critical value (which invalidates an assumption) at time t1; this is observed only after some time (*divergence window Δt*) in components B and C. A respective divergence window again exists in the case when P's value falls back within a "normal" range. It is important to note that data outdatedness does not always lead to divergence, because though different, they can lead to the same decision – this can be observed in Figure 8 in the areas outside the two intervals.

DANC specifically targets systems where temporary divergence of the SAT solving input and, consequently, of its results has only negligible effect in the performance of the system (what is considered "negligible" in this case depends on particular domain requirements). This prerequisite characterizes the type of systems that can employ DANC – when it does not hold, CA or DADC has to be chosen instead.

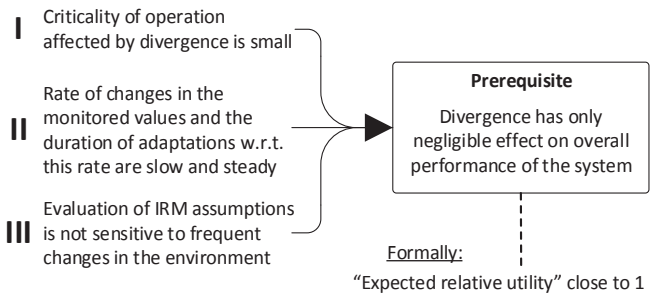


**Figure 9: The three criteria that imply the prerequisite for**

We formalize the above prerequisite and describe the criteria that imply it below. Figure 9 provides a graphical summary.

***Definition 1 (Utility of a system run).*** For a given run of a system:

- Let $a(t)$ be a function that for time $t$ returns the set of active processes as selected by the SAT procedure.
- Let $d(t,k)$ be function that for time $t$ and a knowledge field $k$ returns the time $\Delta t$ that has elapsed since the knowledge value contained in $k$ was sensed/created by the component where the knowledge value originated from.
- Let $a_{c,d}(t)$ be a function that for given time $t$ returns a set of active processes of component $c$ as selected by the SAT procedure assuming that knowledge values outdated by $d(t,k)$ have been used. Further, we denote $a_{d_1,\dots,d_n}(t) = \bigcup_i a_{c_i,d_i}(t)$ as the combination over components existing in the system. (In other words, each component selects active processes itself based on its belief, which is differently outdated for each component.)
- Let $u(a)$ be a cumulative utility function that returns the overall system utility when performing processes $a(t)$ at each time instant $t$.
- Let $u(a,\Delta t_{\max})$ be a cumulative utility function constructed as $\min\{u(a_{d_1,\dots,d_n}) \mid d(k) \le \Delta t_{\max}\}$. (In other words, $u(a,\Delta t_{\max})$ denotes the lowest utility if components decide independently based on knowledge at most $\Delta t_{\max}$ old.)

***Definition 2 (Expected relative utility)***. Let $\mathrm{E}(u(a))$ be the expected value of $u(a)$ and $\mathrm{E}(u(a,\Delta t_{\max}))$ be the expected value of $u(a,\Delta t_{\max})$. Assuming that $\mathrm{E}(u(a)) > 0$ (i.e., in the ideal case of zero communication delays the system provides a positive value), we define expected relative utility as $r(\Delta t_{\max}) = \mathrm{E}(u(a,\Delta t_{\max}))/\mathrm{E}(u(a))$.

We assume systems where $r(\Delta t_{\max})$ is close to 1 (and definitely non-negative) for given upper bound on communication delays $\Delta t_{\max}$. In fact $r(\Delta t_{\max})$ provides a continuous measure of how well the method works in a distributed environment.

Considering that the communication in the system happens periodically and that an arriving message obsoletes all previous not-yet-arrived messages from the same source, $\Delta t_{\max}$ can be set to $q_l + q_d T$, where $q_l$ is a close-to-100% quantile of the distribution of message latencies, $T$ is the period of message sending and $q_d$ is a close-to-100% quantile of the distribution of the length of sequences of consecutive message drops. Naturally, if there is a chance of some (not necessarily reliable communication), $\Delta t_{\max}$ can be set relatively low while still covering the absolute majority of situations.

There are several factors that influence the value of $r(\Delta t_{\max})$. Essentially, it depends on the shape of the utility function (since the utility function is cumulative), on the duration of divergence, and on the robustness of the utility function with respect to divergence. Following this argument, we identify three criteria for the applicability of IRM-SA with DANC. Essentially, "$r(\Delta t_{\max})$ close to 1" is achieved when:

**Criticality of a particular system operation affected by divergence is small**. For critical operations, the utility function tends to get extreme negative values, thus even a short operation under divergence yields very low overall utility. On the other hand, if the environment has revertible and gradual responses, it hardly matters whether the system is in divergence for a limited time (e.g., if the system controls a firefighter walking around a building, then walking in a wrong direction for a few seconds does not cause any harm and its effect can be easily reverted).

**Rate of changes in the monitored values and the duration of adaptations w.r.t. this rate are slow and steady**. As the system reacts to changes in the environment, it is impacted by the speed these changes happen. Such a change creates potential divergence in the system. What matters then is the ratio between the time needed to converge after the divergence and the interval between two consecutive changes. For instance, if house numbers were to be observed by firefighters when passaging by, then walking speed would yield much slower rate of changes then passing by in a car.

**Evaluation of assumptions is not sensitive to frequent changes in the environment**. This is a complementary aspect of the previous property**.** Depending on the way assumptions are refined into computable ones, one can detect fine-grained changes in the environment. For example, consider an assumption that relies on computing position changes of a firefighter moving in a city. Computing position changes based on changes in the house number obviously yields more frequent (observable) changes in the environment than computing changes based on the current street number.

## 7.2 A Case for Decentralized Self-Adaptation with no Distributed Consensus

We now provide a particular scenario where DANC is the most fitting choice, and exemplify the combination of DANC with proactive reasoning.

Consider the scenario where teams of firefighters consisting of three members and one leader were deployed. A firefighter A senses temperature higher than a pre-specified threshold (indication of being "in danger"); this information is propagated to the A's leader who in turn propagates the information that A is in danger to a firefighter B; then, B performs self-adaptation in the anticipation of the harmful situation of having a group member in danger (*proactive self-adaptation*) and switches the mode to "Search and Rescue" (the situation captured by INV-18 in Figure 2). At the point when the leader determines that A is in danger (and just before the leader communicates it to B), a temporary network disconnection occurs. The overall performance was measured by *reaction time* – the interval between the time that A sensed high temperature and the time that B switches to "Search and Rescue". Note that, by setting $u(a)$ to be the inverse of the reaction time, we obtain values for the expected relative utility $r$ that range from $r(1000) = \frac{3000}{8000} = 0.37$ to $r(12000) = \frac{3000}{20000} = 0.15$.

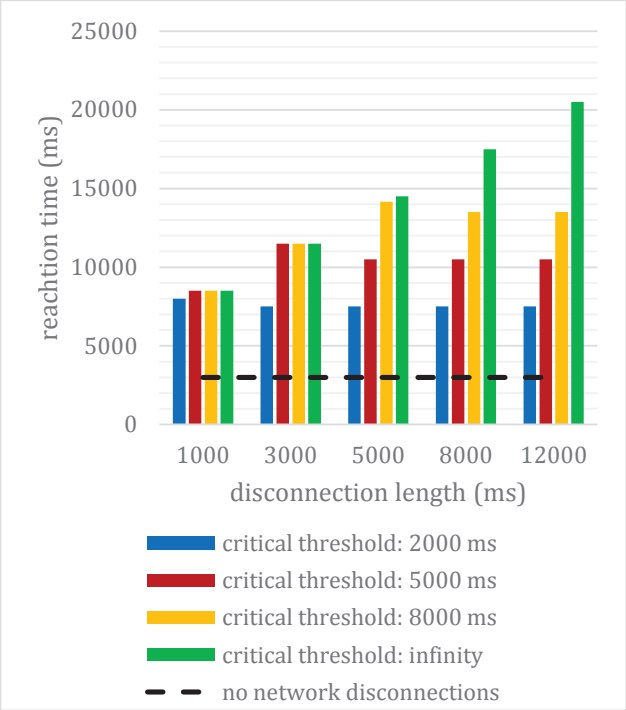


**Figure 10: Reaction times for different network disconnection lengths and different critical thresholds. The results for each case have been averaged for different DEECo component knowledge publishing periods (400, 500 and 600 ms).**

We performed several simulations of the above scenario in order to delineate the limits of proactive self-adaptation. In particular, the questions that we investigated by the experiments were the following.

Q1: *Do temporary network disconnections (and associated communication delays)* ***reduce*** *the overall performance of an application that employs DANC?*

Q2: *Does proactive self-adaptation in IRM-SA method in combination with DANC* ***increase*** *the overall performance of an application in face of temporary disconnections?*

**Simulation setup.** In the experiments we employed an extended version of the running example. This IRM-SA model of consists of 4 components, 39 invariants, and 23 decompositions [42]. The experiments were carried out using the jDEECo simulation engine [44] together with the prototype implementation of IRM-SA in jDEECo (Section 8). Several simulation parameters (such as interleaving of scheduled processes) that were not relevant to our experiment goals were set to fixed values. The simulation code, along with all the parameters and raw measurements, are available online [42].

To obtain a baseline, the case of no network disconnections was also measured. The result is depicted in dashed line in Figure 10.

To investigate Q1 and Q2, a number of network disconnections with preset lengths were considered; this was based on a prior experience of working with deployment of DEECo on mobile ad-hoc networks [14].

To answer Q2, the timed automaton (Figure 3) associated with INV-23: "possibility(GM::nearbyGMsStatus == CRITICAL)" was modified: the transition from DANGER to CRITICAL was made parametric to experiment with different critical threshold values – critical threshold in the context of the experiments is the least time needed for a firefighter to get into a critical situation after he/she gets in danger (in Figure 3 the critical threshold is set to 5 sec). The reaction times for different critical thresholds and different disconnection lengths are in Figure 10.

To answer Q1 (as well as obtain the baseline), the critical threshold was set to infinity – effectively omitting INV-23 from the IRM-SA model – in order to measure the vanilla case where self-adaptation is based only on the values of data transmitted (belief) and not on other parameters such as belief outdatedness and its consequent inaccuracy.

**Analysis of results.** From Figure 10 it is evident that the reaction time (a measure of the overall performance of the system) strongly depends on communication delays caused by temporary disconnections. Specifically, in the vanilla case the performance is inversely proportional to the disconnection length, i.e., it decreases together with the quality of the communication links. This is in favor of a positive answer to Q1.

Also, to cope with operational uncertainty – temporary network disconnections in particular – the IRM-SA mechanisms are indeed providing a solution towards reducing the overall performance loss. Proactive self-adaptation yields smaller reaction times (Figure 10) – this is in favor of a positive answer to Q2. In particular, for the lowest critical threshold (2000ms) the reaction time is fast; this threshold configuration can, however, result into overreactions, since it hardly tolerates any disconnections. When setting the critical threshold to 5000ms, proactive self-adaptation is triggered in case of larger disconnections (5000ms and more) only. A critical threshold of 8000ms triggers proactive self-adaptation in case of even larger disconnections (8000ms or more). Finally, when critical threshold is set to infinity, proactive self-adaptation is not triggered at all.

When disconnection times grow larger and larger, proactive self-adaptation can help in bounding the amount of performance loss. In our case, predictions were based on the timed automaton of Figure 3; in case the possibility of a life-critical situation was predicted, the system switched to the "Search and Rescue" mode. Although several simplifying assumptions were made (e.g., a single, simple automaton was used, all predictions were assumed correct), the experiments provide evidence on the applicability of DANC with proactive self-adaptation within the frame of IRM-SA. Extending the monitoring and prediction capabilities of our framework is subject to future work.

# 8. PROTOTYPE IMPLEMENTATION

We implemented the self-adaptation method of IRM-SA as a plugin (publicly available [42]) into the jDEECo framework [44]. This framework is a realization of the DEECo component model [15, 48] . For developing components and ensembles, jDEECo provides an internal Java DSL and allows their distributed execution. To give a better insight in the prototype implementation, we briefly first overview jDEECo below and then describe the main points of the implementation of IRM-SA.

## 8.1 DEECo Component Model

Dependable Emergent Ensemble of Components (DEECo) is component model (including specification of deployment and runtime computation semantics) tailored for building siCPS with a high degree of dynamicity in their operation. In DEECo, *components* are autonomous units of deployment and computation. Each of them comprises *knowledge* and *processes*. Knowledge is a hierarchical data structure representing the

```
1.  role PositionSensor:
2.    missionID, position
3.
4.  role PositionAggregator:
5.    missionID, positions
6.
7.  component Firefighter42 features PositionSensor, ...:
8.    knowledge:
9.      ID = 42, missionID = 2, position = {51.083582, 17.481073}, ...
10.   process measurePositionGPS (out position):
11.     position ← Sensor.read()
12.   scheduling: periodic( 500ms )
13.   ... /* other process definitions */
14.
15. component Officer13 features PositionAggregator, ...:
16.   knowledge:
17.     ID = 13, missionID = 2, position = {51.078122, 17.485260},
18.       firefightersNearBy = {42, ...},  positions = {{42, {51.083582,
19.         17.481073}},...}
20.   process findFirefightersNearBy(in positions, in position, out
21.     firefightersNearBy):
22.       firefightersNearBy← checkDistance(position, positions)
23.     scheduling: periodic( 1000ms )
24.   ... /* other process definitions */
25. ... /* other component definitions */
26.
27. ensemble PositionUpdate:
28.   coordinator: PositionAggregator
29.   member: PositionSensor
30.   membership:
31.     member.missionID == coordinator.missionID
32.   knowledge exchange:
33.     coordinator.positions ← { (m.ID, m.position) | m ∈ members }
34.   scheduling: periodic( 1000ms )
35. ... /* other ensemble definitions */
```

**Figure 11: Example of possible DEECo components and ensembles in the running example.**

internal state of the component. A process operates upon the knowledge and features cyclic execution based on the concept of feedback loop [60], being thus similar to a process in real-time systems. As an example, consider the two DEECo components in Figure 11, lines 7-13 and 15-23. They also illustrate that separation of concerns is brought to such extent that individual components do not explicitly communicate with each other. Instead, interaction among components is determined by their composition into *ensembles* – groups of components cooperating to achieve a particular goal [23, 40] (e.g., PositionUpdate ensemble in Figure 11, lines 27-34). Ensembles are dynamically established/disbanded based on the state of components and external situation (e.g., when a group of firefighters are physically close together, they form an ensemble). At runtime, *knowledge exchange* is performed between the components within an ensemble (lines 32-33) – essentially updating their beliefs (Section 3).

## 8.2 jDEECo Runtime

Each node in a jDEECo application contains a jDEECo runtime, which in turn contains one or more local components – serving as a container (Figure 13). The runtime is responsible for periodical scheduling of component processes and knowledge exchange functions (lines 12, 23, 34). It also possesses reflective capabilities in the form of a *DEECo runtime model* that provides runtime support for dynamic reconfigurations (e.g., starting and stopping of process scheduling). Each runtime manages the knowledge of both *local components*, i.e., components deployed on the same node, and *replicas,* i.e., copies of knowledge of the components that are deployed on different nodes but interact with the local components via ensembles.

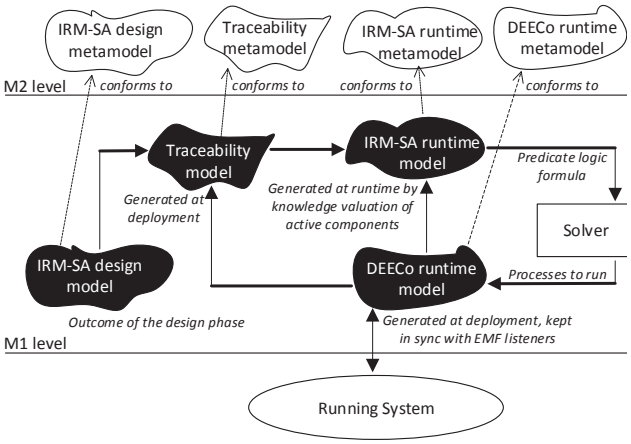


**Figure 12: Models and their meta-models employed for self-adaptation in jDEECo.**

## 8.3 Self-Adaptation in jDEECo

For integration of the self-adaptation method of IRM-SA *(jDEECo self-adaptation*) with jDEECo, a models-at-runtime approach [57] is employed, leveraging on EMF-based models (Figure 12). In particular, a *Traceability model* is created at deployment, providing the association of entities of the DEECo runtime model (i.e., components, component processes, and ensembles) with the corresponding constructs of the *IRM-SA design model* (i.e., components and invariants). This allows traceability between entities of requirements, design, and implementation – a feature essential for self-adaptation. For example, the process measurePositionGPS in Figure 11 is traced back to INV-27 (line 7), while the Firefighter component is traced back to its IRM-SA counterpart (line 2). Based on the Traceability model and the DEECo runtime model, an *IRM-SA runtime model* is generated by "instantiating" the IRM-SA design components with local components and replicas. Once the IRM-SA runtime model gets used for selecting an architecture configuration, the selected configuration is imposed to the DEECo runtime model as the current one.

A central role in performing jDEECo self-adaptation is played by a specialized jDEECo component Adaptation Manager (AM). Its functionality comprises the following steps (Figure 13): (1) Aggregation of monitoring results from local components and replicas and creation of IRM-SA runtime model. (2) Invocation of the SAT solver (Sections 6.1-6.3). (3) Translation of the SAT solver output into an applicable configuration (including prioritization). (4) Triggering the actual architecture adaptation – applying the current configuration. As an aside, internally, AM employs the SAT4J solver [53], mainly due to its seamless integration with Java.

The essence of step (4) lies in instrumenting the scheduler of the jDEECo runtime. Specifically, for every process, resp. exchange invariant in the current configuration, AM starts/resumes the scheduling of the associated component process, resp. knowledge exchange function. The other processes and knowledge exchange functions are not scheduled any more.

## 8.4 Monitoring

AM can handle both active and predictive monitoring techniques (Section 6.3). In the experiments described in Section 7.2, predictive monitoring was used for both component processes and knowledge exchange functions (based on observing the history of

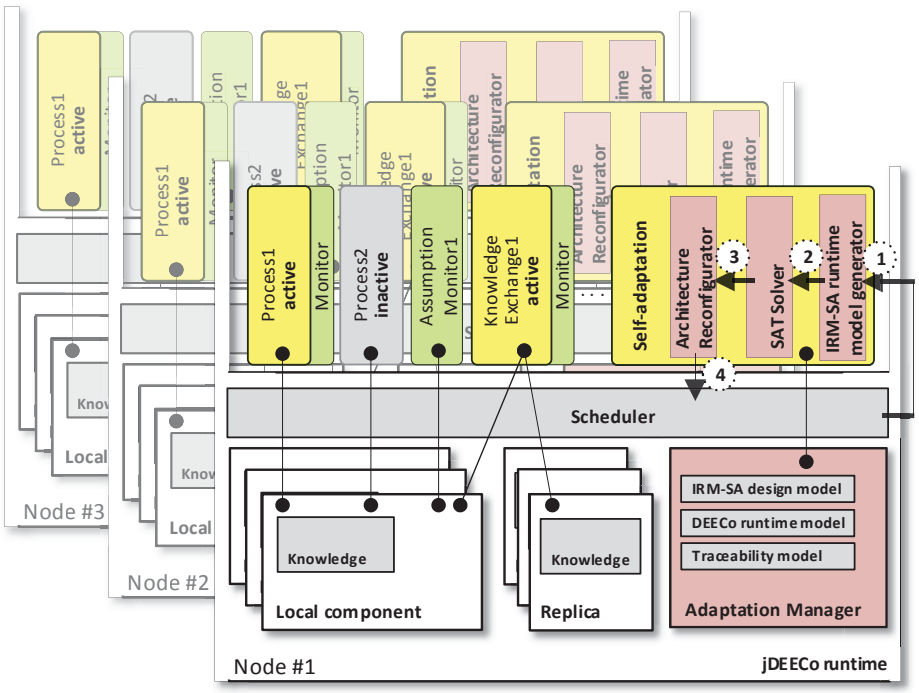


**Figure 13: Steps in jDEECo self-adaptation: (1) Aggregate monitoring results from local component and replicas and create IRM-SA runtime model; (2) Translate into SAT formula, bind monitoring variables; (3) Translate SAT solver output to current configuration; (4) Activate/ deactivate component processes and knowledge exchange processes according to current configuration. At each node, self-adaptation is a periodically-invoked process of the "Adaptation Manager" system component (which is deployed on each node along with the application components).**

invariants evaluation), while for assumptions, only active monitoring was employed.

Technically, monitors are realized as Boolean methods associated with invariants in the Traceability model. For instance, a monitor for INV-27: "GM::position is determined from GPS" is illustrated in Figure 14, lines 16-22; it checks whether the corresponding process operates correctly by checking the health of the GPS device and the number of available satellites. The execution of monitors is driven directly by AM and is part of the first step of the jDEECo self-adaptation (Figure 13).

# 9. IRM-SA MODELING: FEASIBILITY AND EFFECTIVENESS

To evaluate the feasibility of the IRM-SA modeling process, described in Section 5.3, and the impact of using IRM-SA on the effectiveness of the architects, we carried out an empirical study in the form of a controlled experiment.

## 9.1 Experiment Design

The experiment involved 20 participants: 12 Master's and 8 Ph.D. students of computer science. The participants were split into treatment (IRM-SA) and control groups. Each participant was assigned the same task, which involved coming up with a specification (on paper) of system architecture comprised of DEECo components and ensembles for a small system-to-be. The requirements of the system-to-be were provided in the form of user stories. The task's effort was comparable in size of invariants, situations and decompositions to the running example in this paper. The suggested time to accomplish the task was 4 hours, although no strict limit was imposed.

The independent variable/main factor of the experiment was the design method used: Participants in the IRM-SA group followed the IRM-SA modeling process to come up with an IRM-SA model, and then manually translated it to the system architecture, whereas participants in the control group were not recommended to use any specific method for designing the system architecture.

The dependent variables of the experiment are mapped to the hypotheses in Tables 1 and 2. We assessed (i) the correctness of system architecture [0-100] ratio scale, and (ii) several other variables capturing the intuitiveness of IRM-SA, perceived effort, adequateness of the experiment settings, etc. in Likert 5-

```
1.  @Component
2.  @IRMComponent("Firefighter")
3.  public class Firefighter {
4.    public Position position;
5.    ...
6.
7.    @Invariant("27")
8.    @PeriodicScheduling(period=500)
9.    public static void monitorPositionGPS(
10.     @Out("position") Position position
11.   ) {
12.     // read current position from the GPS device
13.   }
14.   ...
15.
16.   @InvariantMonitor("27")
17.   public static boolean monitorPositionGPSMonitor(
18.     @In("position") Position position
19.   ) {
20.     // check health of GPS device
21.     // check if at least 3 satellites are visible
22.   }
23.   ...
24. }
```

**Figure 14: Firefighter component definition and process-monitor definition in the internal Java DSL of jDEECo.**

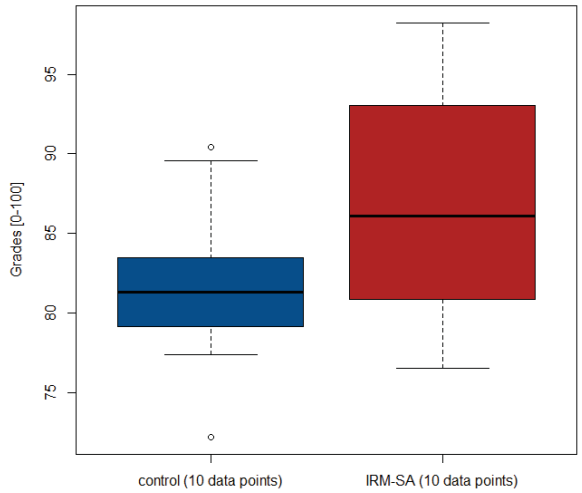


**Figure 15: Box-and-whisker diagrams of the two samples measuring correctness of system architectures.**

point ordinal scale (1: Strongly disagree, 2: Disagree, 3: Not certain, 4: Agree, 5: Strongly agree). While (i) was based on manual grading each architecture based on a strict grading protocol where each error (missing knowledge, missing role, wrong process condition, wrong process period, wrong ensemble membership condition, etc.) was penalized by reducing certain number of points, assessment of (ii) was provided by the participants in pre- and post-questionnaires.

## 9.2 Results and Interpretation

To analyze the results and draw conclusions, for each hypothesis, we formulated a null hypothesis and ran a one-sided statistical test to reject it. In the case of two-sample tests (Table 1), the null hypothesis stated that the medians of the two groups were significantly different, whereas in the case of one-sample tests (Table 2) it stated that the answer is not significantly more positive than "Not certain" (point 3 in Likert). We adopted a 5% significance level, accepting the null hypothesis if p-value<0.05. (The p-value denotes the lowest possible significance with which it is possible to reject the null hypothesis [78]).

The correctness of the system architectures showed a statistically significant difference (with p=0.0467) in favor of the IRM-SA group (Table 1, H1). Figure 15 depicts the dispersion of grades around the medians for the two groups, while Figure 16 depicts the frequencies of grades per group. From H9 (Table 2) we can also conclude that participants of the IRM-SA group perceived IRM-SA as an important factor of their confidence on the correctness of the system architecture they proposed. These two results allow us to conclude that IRM-SA increased both the actual and the perceived effectiveness of the modeling process to a statistically significant extent.

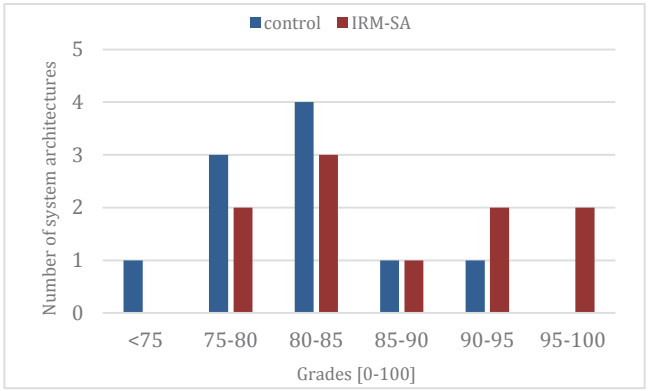


**Figure 16: Histogram with distribution of grades per group.**

Regarding the rest of the conclusive results, participants of the IRM-SA group found the IRM-SA modeling process intuitive (H4) and the IRM-SA concepts rich enough to capture design choices (H6).

## 9.3 Threats to Validity

**Conclusion validity.** To perform the parametric t-test for interval/ratio data for H1, we assumed normal distribution of samples [71]. Since Likert data were treated as ordinal, we used the non-parametric Mann-Whitney tests for H2-H3 and Wilcoxon tests for H4, H6 and H9 [71]. Grading was based on a strict pre-defined protocol, which we made available together with the anonymized raw data and the replication packages in the experiment kit [42].

**Internal validity.** We adopted a simple "one factor with two treatments" design [78] to avoid learning effects. The number of participants (20) was high enough to reach a basic statistical validity. We used a semi-randomized assignment of participants

**Table 1: Two-sample tests to compare the populations of IRM-SA and control groups.**

| id | Alternative (Null) Hypothesis | test | median control | median IRM-SA | reject null-H? | p-value |
|---|---|---|---|---|---|---|
| 1 | The correctness of the final system architectures is lower (null: the same) in the control group than in the IRM-SA group | t-test | 81.30 | 86.09 | **Y** | 0.0467 |
| 2 | The IRM-SA group witnessed less (null: the same) difficulties in coming up with a DEECo architecture than the control group | Mann-Whitney | 4 | 4 | N | 0.5164 |
| 3 | The IRM-SA group perceived the design effort as more (null: equally) likely to be too high for an efficient use of the methodology in practice | Mann-Whitney | 2 | 2.5 | N | 0.4361 |

**Table 2: One-sample Wilcoxon Signed-Rank tests to assess a hypothesis specific to a group. Only conclusive results are shown; for the complete list of results we refer the interested reader to the experiment kit [42].**

| id | Alternative Hypothesis | median | p-value |
|---|---|---|---|
| 4 | IRM-SA group will find it easy to think of a system in IRM-SA concepts (invariants, assumptions) | 3.5 | 0.02386 |
| 6 | IRM-SA group will find IRM-SA concepts detailed enough to captured the design choices | 4 | 0.00176 |
| 9 | IRM-SA group will have increased confidence over the correctness of the architecture via IRM-SA | 4 | 0.00731 |

to groups so that each group is balanced both in terms of Master's vs Ph.D. students and in terms of their experience with DEECo. Although the average time to completion of the assignment varied greatly, the mean (140 mins) and minimum (75 mins) values indicate that participants spent enough time to understand, think about and perform the task and fill in the questionnaires. The material and the experiment process was beta-tested with 4 participants beforehand.

**Construct validity.** We dealt with mono-method bias by using both subjective (questionnaires) and objective (grades) measures. We also measured more constructs than needed for H1 in order to obtain multiple sources of evidence. However, we introduced mono-operation bias by using only one treatment (IRM-SA) and one task. Mitigating this threat is subject of additional future studies.

**External validity.** Our population can be categorized as junior software engineers, since it was formed by graduate students in the last years of their studies [41]. Nevertheless, we used a rather simple example and no group assignments. Therefore, the results can be generalized mainly in academic settings.

# 10. DISCUSSION

In this section, we reflect on our experience with designing with IRM-SA and discuss the ways to cope with unanticipated situations in IRM-SA (Section 10.1). We also delineate the applicability of IRM-SA beyond DEECo (Section 10.2) and provide a critical review of the contributions of our work (Section 10.3).

## 10.1 Coping with Unanticipated Situations

Although IRM-SA modeling and self-adaptation as described in the previous sections relies on anticipated situations, we are aware of the fact that siCPS often need to operate in situations that reside out of their "envelope of adaptability" [10]. In this section, we explain how IRM-SA tackles this problem both at runtime and design time. The driving idea is to control the decline of dependability in the system caused by unanticipated situations, so that the system's operations degrade gradually in a controlled manner.

To illustrate the problem using the running example, consider a scenario of a vegetation fire where firefighters, as a part of coordinating their actions, periodically update their group leader with information about their position as captured by personal GPS devices. A problem arises when GPS monitoring fails for whatever reasons (battery drainage, etc.) – the system would no longer be able to adapt, since this failure was not anticipated at the design time. Below, we explain several strategies we propose to cope with such unanticipated situations in IRM-SA.

### *10.1.1 Runtime Strategy*

The principal strategy for coping with unanticipated situations is to specify alternatives of OR-decompositions in such a way that they cover situations in an overlapping manner. This increases overall system robustness and fault-tolerance by providing a number of alternatives to be selected in a particular situation.

A special case is when an IRM-SA model contains one or more design alternatives that have very weak assumptions, i.e., assumptions that are very easily satisfied at runtime, and minimum preference in an OR-decomposition. Such a design alternative is chosen only as the last option as a fail-safe mode, typical for the design of safety-critical systems. Figure 17 (a) depicts such a case, where the situation 3, reflecting a fail-safe mode, overlaps with both the situation 1 and 2.

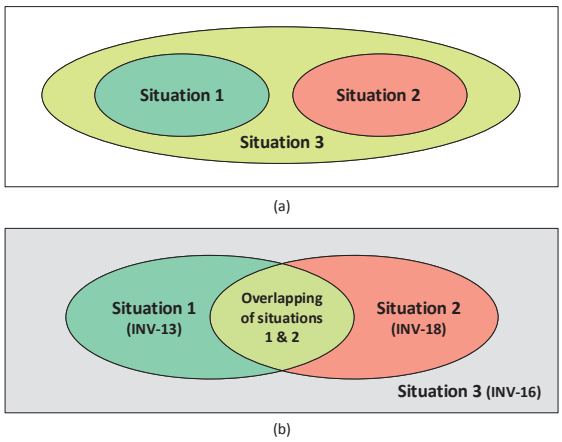


**Figure 17: Overlap of situations when (a) situation 3 is a "fail-safe" mode, (b) situations overlap in the running example.**

In the running example, the runtime strategy is employed in two OR-decompositions (Figure 2). The left-most part of the decomposition of INV-9 "GM::sensorData is determined" has to be maintained differently when the associated group member is in danger (INV-13), when a nearby group member is in danger/critical state (INV-18), and when no life is in threat (INV-16). The situation characterized by INV-16 stands as a counterpart of the other two, nevertheless they are not mutually exclusive. This case is depicted in Figure 17.b.

Further, the INV-12: "GM::position is determined" has to be maintained in the two situations characterized by the INV-24: "GM indoors" and INV-26: "GM outdoors". The last two also potentially overlap, corresponding to the real-life scenario where a firefighter repeatedly enters and exits a building. In this case, the firefighter can also use the indoors tracking system to track his position; this design alternative is automatically chosen when the GPS unexpectedly malfunctions.

### *10.1.2 Re-design Strategy*

The re-design strategy is applied in the design evolution process – occurrences of the adaptation actions that led to a failure in the system are analyzed and the IRM-SA model is revised accordingly. Such a revision can range from inclusion of a single invariant to restructuring of the whole IRM-SA model.

In such a revision, an important aid for the designer is the fact that each invariant refinement implies relationship between the sub-invariants and the parent invariant (Section 5.1). By monitoring the satisfaction of the parent invariant $I_p$ and sub-invariants $I_1, \dots, I_n$, it is possible to narrow down the adaptation failure and infer a suitable way of addressing it. In particular, an adaptation failure occurs when:

(a) $I_p$ is AND-decomposed, all non-process invariants among $I_1, \dots, I_n$ hold but $I_p$ does not hold. This points to a concealed assumption in the refinement of $I_p$.

(b) $I_p$ is OR-decomposed, none of its alternatives holds, but $I_p$ holds. This points to the fact that the refinement of $I_p$ is likely to have more strict assumptions than necessary.

(c) $I_p$ is OR-decomposed, none of its alternatives holds, and $I_p$ does not hold as well. This points to such an unanticipated situation, which requires either a new alternative to be introduced or an alternative that provides "close" results to be extended.

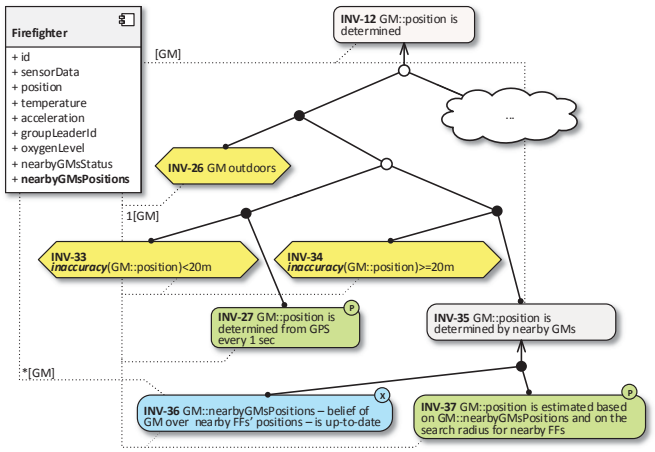


**Figure 18: Design evolution scenario – the new "*inaccuracy*(GM::position) < 20m" situation is added to the model and drives the adaptation.**

For illustration (of the case (c) in particular), consider the scenario of a non-responsive GPS. In this case, both "GM::position is determined from GPS every 1 sec" (INV-27 in Figure 2) and its parent "GM::position is determined" (INV-12) do not hold, which is a symptom for an unanticipated situation. Indeed, the root cause is that GPS signal was considered accurate. To mitigate this problem, we employ the evolution of the running example as presented in Figure 18. There, the unanticipated situation has become explicit and is used to drive the adaptation. Specifically, in this new situation, the system still satisfies INV-12 by switching to the right-most alternative. In such a case, the Firefighter's position is determined by aggregating the positions of the nearby firefighters (INV-36) and estimating its own position based on these positions and the radius of search, through determining the maximum overlapping area (INV-37).

## 10.2 Applicability of IRM-SA beyond DEECo

The IRM-SA concepts of design components and invariants, and the process of elaborating invariants via AND- and OR-decomposition are implementation-agnostic. The outcome therefore of the design – the IRM-SA model – can be mapped to a general-purpose (e.g., OSGi [34], Fractal [13], SOFA 2 [16]) or real-time component model (e.g., RTT [75], ProCom [68]) that provides explicit support for modeling entities that encapsulate computation (components) and interaction (connectors). In a simple case, the mapping involves translating the process and exchange invariants to local component activities and to connectors, respectively.

That said, the mapping of IRM-SA to DEECo is particularly smooth. This is because IRM-SA is tailored to the domain of dynamic self-adaptive siCPS, where the management of belief of individual components via soft-real-time operation and dynamic grouping has a central role. In this regard, component models tailored to self-adaptive siCPS (e.g., DEECo) and architectural styles for group-based coordination (e.g., A-3 [4]), are good candidates for a smooth mapping, since they provide direct implementation-level counterparts of the IRM-SA abstractions.

## 10.3 Limits of Contributions

We now critically review each of the four key contributions of this paper, as articulated in Section 1.1.

With respect to (i), modeling with IRM-SA has been shown via a controlled experiment to be in principle feasible and effective for the design of software for siCPS. However, the running example, the system used in our experiments with jDEECo, and the system the participants modeled in the empirical study were all of considerably small size. A question remains as to how the modeling approach scales in terms of the requirements and situations that need to be elaborated and considered at design time. A key to this is to support the modeling task with appropriate tooling, such as graphical editors with advanced features (e.g., allowing for IRM-SA subtrees to be referenced at different nodes in the graph).

With respect to (ii), the automated self-adaptation method relies on SAT solving to find an applicable configuration, and therefore potentially hindered by the computational limitations of SAT solvers. Nevertheless, contemporary solvers can easily scale to several thousands of variables and clauses [7]. (In our experiments (Section 7.2), the SAT instances handled were in the order of hundreds (approx. 100 variables and 200 clauses)). When using DANC, we assume that the IRM-SA reasoning will be localized to groups of connected nodes, so that the resulting SAT instances will be manageable.

With respect to (iii), we provided a set of experiments showing how the DANC implementation of IRM-SA deals with network disconnections in specific, but non-trivial settings. The lesson is that DANC can be employed specifically in a class of "well-behaved" systems, as detailed in Section 7.1. When the siCPS in question does not belong to this class, techniques from networked control systems [33] resp. their wireless variants [36], and multi-agent coordination [43, 62] have to be sought out.

With respect to (iv), we provided two strategies to deal with unanticipated situations; the first one refers primarily to designing with fail-safe modes – a strategy widely used in CPS [70]; the second one refers to error detection and principled design evolution that take advantage of the position of assumptions in the IRM-SA graph. We acknowledge that these strategies do not cope with fully unanticipated situations and environmental uncertainty. In our current and future work, we are focusing on devising new architecture alternatives at runtime to cope with such cases [31].

Finally, there are several difficult research topics related to control theory [37], such as timeliness of adaptation triggering, sequential decision-making issues, and timing issues related to observing the effects of adaptations on the monitored values in order to plan further adaptations. Nevertheless, these issues lie out of the scope of this paper.

# 11. RELATED WORK

Recently, there has been a growing interest in software engineering research for software-intensive systems with self-adaptive and autonomic capabilities [65]. Since our approach is especially close to software product lines approaches and their dynamic counterparts, we first position IRM-SA against these approaches; we then split the comparison into three essential views of self-adaptation [18], namely requirements, assurances, and engineering.

## 11.1 IRM-SA vs. Dynamic Software Product Lines

Variability modeling at design time has been in the core of research in software product lines (SPLs) [21]. Recently, numerous approaches have proposed to apply the proven SPL techniques of managing variability in the area of runtime

adaptation, leading to the concept of dynamic SPL (DSPL) [38]. The idea behind DSPL is to delay the binding of variants from design to runtime, to focus on a single product than a family of products, and to have configurations be driven by changes in the environment than by market needs and stakeholders. In this spirit, IRM-SA can be considered as a DSPL instance.

Within DSPLs, feature models are an effective means to model the common features, the variants, and the variability points of a self-adapting system [45]. However, they fall short in identifying and modeling the situations that the system may encounter during operation [67]. This results in maintaining an additional artifact as a context model (e.g., an ontology [59]) in parallel with the feature model at runtime, in order to bind the monitoring and planning phases of the MAPE-K loop [46]. IRM-SA provides an explicit support for capturing situations, pertinent to system operation and self-adaptation, via the assumption concept. At the same time, by building on the iterative refinement of invariants and the assignment of leaf invariants to design components, IRM-SA integrates the problem space view (requirements models) with the solution space view (feature models) into a single manageable artifact (IRM-SA model).

Finally, compared to existing approaches in DSPLs that use goal models to describe the configuration space [67], IRM-SA does not focus on the prioritization between applicable configurations, but on the gradual degradation of overall system performance when dealing with operational uncertainty in decentralized settings.

## 11.2 Requirements and Assurances

In an effort to study the requirements that lead to the feedback loop functionality of adaptive systems, Souza et al. defined a new class of requirements, termed "awareness requirements" [73], which refer to the runtime success/failure/quality-of-service of other requirements (or domain assumptions). Awareness requirements are expressed in an OCL-like language, based on the Tropos goal models [12] produced at design time, and monitored at runtime by the requirements monitoring framework ReqMon [64]. The idea is to have a highly sophisticated logging framework, on top of which a full MAPE-K feedback loop [46] can be instantiated. Our approach, on the other hand, features a tighter coupling between monitoring and actuating, since both aspects are captured in the IRM-SA model.

Extending goal-based requirements models with alternative decompositions to achieve self-adaptivity has been carried out in the frame of Tropos4AS [55, 56]. System agents, together with their goals and their environment are first modeled and then mapped to agent-based implementation in Jadex platform. Although IRM-SA is technically similar to Tropos4AS, it does not capture the goals and intentions of individual actors, but the desired operation of the system as a whole, thus promoting dependable operation, key factor in siCPS.

For dealing with uncertainty in the requirements of self-adaptive systems, the RELAX [77] and FLAGS [5] languages have been proposed. RELAX syntax is in the form of structured natural language with Boolean expressions; its semantics is defined in a fuzzy temporal logic. The RELAX approach distinguishes between invariant and non-invariant requirements, i.e., requirements that may not have to be satisfied at all times. In [19], RELAX specifications are integrated into KAOS goal models. Threat modeling à la KAOS [52] is employed to systematically explore (and subsequently mitigate) uncertainty factors that may impact the requirements of a DAS. FLAGS is an extension to the classical KAOS goal model that features both crisp goals (whose satisfaction is Boolean) and fuzzy goals (whose satisfaction is represented by fuzzy constrains) that correspond to invariant and non-invariant requirements of RELAX. The idea of using a FLAGS model at runtime to guide system adaptation has been briefly sketched in [5], but to the best of our knowledge not yet pursued. Compared to IRM-SA, both RELAX and FLAGS focus on the requirements domain and do not go beyond goals and requirements to design and implementation.

One of the first attempts to bind requirements, captured in KAOS, with runtime monitoring and reconciliation tactics is found in the seminal work of Fickas and Feather [25, 26]. Their approach is based on capturing alternative designs and their underlying assumptions via goal decomposition, translating them into runtime monitors, and using them to enact runtime adaptation. Specifically, breakable KAOS assumptions (captured in LTL) are translated into the FLEA language, which provides constructs for expressing a temporal combination of events. When requirements violation events occur, corrective actions apply, taking the form of either parameter tuning or shifting to alternative designs. There are two main differences to our approach: (i) in KAOS-FLEA the designer has to manually write and tune the reconciliation tactics, whereas we rely on the structure of an IRM-SA model and the solver for a solution; (ii) contrary to KAOS-FLEA we do not treat alternative designs as correcting measures, but as different system modes, which is more suitable for siCPS environments.

## 11.3 Design and Implementation

At the system design and implementation phases, the component-based architectural approach towards self-adaptivity is favored in several works [29, 63]. Here, mode switching stands as a widely accepted enabling mechanism, introduced by Kramer and Magee [39, 49]. The main shortcoming is that mode switching is triggered via a finite state machine with pre-defined triggering conditions, which is difficult to trace back to system requirements. Also, although partially addressed in [58], modes and the triggering conditions are usually designed via explicit enumeration, which may cause scalability problems given the number and complex mutual relations of the variation points involved. In IRM-SA, architecture configurations act as modes. However, IRM-SA complements mode switching by enabling for compositional definition of architecture configurations and providing the traceability links, which in turn allows for self-explanation [66].

Another large area of related work focusing specifically on decentralized operation to achieve a common joint goal is found in distributed and multi-agent planning. Here the main questions are centered around the issue of how to compute a close-to-optimal plan for decentralized operation of agents with partially observable domains. The typical solution is to model the environment of an agent via a Decentralized Partially Observable Markov Decision Process (DEC-POMDP) [8]. This is further extended to include imperfect communication [74, 80] and taken even a step further, when decentralization is strengthened by not requiring prior coordination (and strict synchronization) – either by resigning on the inter-agent communication [6, 79] or by performing communication simultaneously to planning [76]. A disadvantage is that decentralized approaches to MDP do not scale beyond a few number of agents [6, 76] – standard benchmark problems in the DEC-POMDP community typically involve just two agents (e.g., [2, 9, 69]). The major difference to our work is that multi-agent planning requires a model of the environment in order to predict the effect of an action. A complete model of an environment becomes a rather infeasible requirement. IRM-SA does not require the complete model of an environment

as the plan is not computed, but specified by a designer. By including high-level invariants, the IRM-SA however still able to reason about the efficiency of the configuration being currently executed and it also drives decentralized decisions on the configuration selection.

Finally, architecture adaptation based on various constraint solving techniques is not a new idea. A common conceptualization, e.g., in [30, 35], is based on the formal definition of architecture constraints (e.g., architecture styles, extra-functional properties), individual architecture elements (e.g., components), and, most importantly, adaptation operations that are supported by an underlying mechanism (e.g., addition/removal of a component binding). Typically, the objective is to find an adaptation operation that would bring the architecture from an invalid state to a state that conforms to the architecture constraints (architecture planning). Although these methods support potentially unbounded architecture evolution (since they are based on the supported adaptation operations rather than predefined architecture configurations), they typically consider only structural and extra-functional properties rather than system goals. Consequently, they support neither smooth, gradual degradation in unanticipated situations, nor offline evolution of the design.

# 12. CONCLUSION AND FUTURE WORK

In this paper, we have presented the IRM-SA method – an extension to IRM that allows designing of self-adaptive software-intensive Cyber-Physical Systems (siCPS) with a focus on dependability aspects. The core idea of the method is to describe variability of a system by alternative invariant decompositions and then to drive system adaptation by employing the knowledge of high-level system's goals and their refinement to computational activities.

A novel feature of IRM-SA is that it allows adaptation in presence of operational uncertainty caused by inaccuracies of sensed data and by unreliable communication. This is achieved by reasoning not only on the values of belief on component knowledge, but also on the degree of belief inaccuracy, which is based on the degree of belief outdatedness and the model that governs the evolution of belief.

As a proof of concept, we have implemented IRM-SA within jDEECo (a Java realization of the DEECo component model) and showed its feasibility by modeling a real-life scenario from an emergency coordination case study. Moreover, we have provided a proof-of-concept demonstration of the benefits of IRM-SA by experiments carried out by simulation. We have also tested the feasibility and effectiveness of the IRM-SA modeling process via a controlled experiment.

**Future work**. When the SAT solver fails to find an applicable configuration, self-adaptation based on IRM-SA reaches its limits. In our current work we aim at extending the envelope of adaptability of architecture-based self-adaptive systems. We do so by introducing new architecture alternatives (*tactics*) to the running system when no applicable IRM_SA configuration exists [31]. For example, one of the ways of introducing new alternatives exploits the fact that sensed data are often correlated (e.g. in siCPS, certain measurable attributes are typically location-dependent), which creates the possibility of employing "collaborative sensing" when direct (local) sensing is not possible.

Another promising idea is to build on the fact that, in siCPS physical laws govern the evolution of the values of specific physical-environment magnitudes (e.g. speed, position). Such laws can be employed in predicting such evolution. We have already introduced the *inaccuracy* construct (Section 5.2) to integrate the results of such predictions with IRM-SA. This idea aligns well with research in stochastic hybrid systems, and also complies with our vision to employ Multi-Paradigm Modeling (MPM) in siCPS [61]. MPM that will allow to work with different types of models (such as physical-world, communication, mechanical, and adaptability models) which would be integrated into a single rich software architecture model (similar to [11]). For instance, this would significantly help in cases when new alternatives in the adaptability model are to be introduced while being subject to safety constraints in the physical-world model.

Finally, focusing on the important distributed control aspects of siCPS and modeling them in software-engineering models such as IRM-SA is another interesting topic for future work.

# 13. ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers for their constructive and detailed feedback on earlier versions of this manuscript.

This work was partially supported by the EU project ASCENS 257414 and by Charles University institutional funding SVV-2015-260222. The research leading to these results has received funding from the European Union Seventh Framework Programme FP7-PEOPLE-2010-ITN under grant agreement n°264840. This work was partially supported by the project no. LD15051 from COST CZ (LD) programme by the Ministry of Education, Youth and Sports of the Czech Republic.

# 14. REFERENCES

[1] Ali, R. Al, Bures, T., Gerostathopoulos, I., Keznikl, J. and Plasil, F. 2014. Architecture Adaptation Based on Belief Inaccuracy Estimation. *Proc. of WICSA'14* (Sydney, Australia, Apr. 2014), 1–4.

[2] Amato, C. and Zilberstein, S. 2009. Achieving Goals in Decentralized POMDPs. *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems* (Richland, SC, 2009), 593–600.

[3] Arcaini, P., Gargantini, A. and Vavassori, P. 2015. Generating Tests for Detecting Faults in Feature Models. *Proc. of the 8th IEEE International Conference on Software Testing, Verification and Validation (ICST '15), to appear.* (Apr. 2015).

[4] Baresi, L. and Guinea, S. 2011. A-3: An Architectural Style for Coordinating Distributed Components. *2011 9th Working IEEE/IFIP Conference on Software Architecture (WICSA)* (Jun. 2011), 161–170.

[5] Baresi, L., Pasquale, L. and Spoletini, P. 2010. Fuzzy Goals for Requirements-Driven Adaptation. *2010 18th IEEE International Requirements Engineering Conference*. (Sep. 2010), 125–134.

[6] Barrett, S., Stone, P., Kraus, S. and Rosenfeld, A. 2013. Teamwork with Limited Knowledge of Teammates. *Proceedings of the 27th AAAI Conference on Artificial Intelligence* (Jul. 2013).

[7] Belov, A., Diepold, D., Heule, M.J.H. and Järvisalo, M. 2014. Proceedings of SAT Competition 2014: Solver and Benchmark Descriptions. (2014).

[8] Bernstein, D.S., Givan, R., Immerman, N. and Zilberstein, S. 2002. The Complexity of Decentralized Control of Markov Decision Processes. *Math. Oper. Res.* 27, 4 (Nov. 2002), 819–840.

[9] Bernstein, D.S., Hansen, E.A. and Zilberstein, S. 2005. Bounded Policy Iteration for Decentralized POMDPs. *Proceedings of the 19th International Joint Conference on Artificial Intelligence* (San Francisco, CA, USA, 2005), 1287–1292.

[10] Berry, D.M., Cheng, B.H.C. and Zhang, J. 2005. The Four Levels of Requirements Engineering for and in Dynamic Adaptive Systems. *Proc. of the 11th International Workshop on Requirements Engineering Foundation for Software Quality, Porto, Portugal* (2005), 95–100.

[11] Bhave, A., Krogh, B.H., Garlan, D. and Schmerl, B. 2011. View Consistency in Architectures for Cyber-Physical Systems. *2011 IEEE/ACM International Conference on Cyber-Physical Systems (ICCPS)* (Apr. 2011), 151–160.

[12] Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F. and Mylopoulos, J. 2004. Tropos: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems*. 8, 3 (May 2004), 203–236.

[13] Bruneton, E., Coupaye, T., Leclercq, M., Quema, V. and Stefani, J.-B. 2006. The Fractal component model and its support in Java. *Software: Practice & Experience*. 36, 11-12 (2006), 1257–1284.

[14] Bures, T., Gerostathopoulos, I., Hnetynka, P. and Keznikl, J. 2014. Gossiping Components for Cyber-Physical Systems. *Proc. of 8th European Conference on Software Architecture* (2014), 250–266.

[15] Bures, T., Gerostathopoulos, I., Hnetynka, P., Keznikl, J., Kit, M. and Plasil, F. 2013. DEECo – an Ensemble-Based Component System. *Proc. of CBSE'13* (Vancouver, Canada, Jun. 2013), 81–90.

[16] Bures, T., Hnetynka, P. and Plasil, F. 2006. SOFA 2.0 : Balancing Advanced Features in a Hierarchical Component Model. *SERA '06* (2006), 40–48.

[17] Chang, N., Rashidzadeh, R. and Ahmadi, M. 2010. Robust indoor positioning using differential wi-fi access points. *IEEE Transactions on Consumer Electronics*. 56, 3 (Aug. 2010), 1860–1867.

[18] Cheng, B. et al. 2009. Software Engineering for Self-Adaptive Systems: A Research Roadmap. *Software Engineering for Self-Adaptive Systems*. Springer Berlin Heidelberg. 1–26.

[19] Cheng, B.H.C., Sawyer, P., Bencomo, N. and Whittle, J. 2009. A Goal-Based Modeling Approach to Develop Requirements of an Adaptive System with Environmental Uncertainty. *Proc. of the 12th International Conference on Model Driven Engineering Languages and Systems, MoDELS '09* (2009), 1–15.

[20] Chung, L., Nixon, B., Yu, E. and Mylopoulos, J. 1999. *Non-Functional Requirements in Software Engineering*. Springer.

[21] Clements, P. and Northrop, L. 2002. *Software Product Lines: Practices and Patterns*. Addison Wesley Professional.

[22] Dalpiaz, F., Chopra, A.K., Giorgini, P. and Mylopoulos, J. 2010. Adaptation in Open Systems: Giving Interaction its Rightful Place. *Proceedings of the 29th International Conference on Conceptual Modeling (ER '10)* (Vancouver, Canada, Nov. 2010), 31–45.

[23] DeNicola, R., Ferrari, G., Loreti, M. and Pugliese, R. 2013. A Language-based Approach to Autonomic Computing. *Formal Methods for Components and Objects*. 7542, (2013), 25–48.

[24] Esfahani, N., Kouroshfar, E. and Malek, S. 2011. Taming Uncertainty in Self-Adaptive Software. *SIGSOFT/FSE '11: Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering* (2011), 234–244.

[25] Feather, M.S., Fickas, S., van Lamsweerde, A. and Ponsard, C. 1998. Reconciling System Requirements and Runtime Behavior. *Proceedings of the 9th International Workshop on Software Specification and Design* (1998), 50–59.

[26] Fickas, S. and Feather, M.S. 1995. Requirements monitoring in dynamic environments. *Proceedings of 1995 IEEE International Symposium on Requirements Engineering (RE'95)*. (1995), 140–147.

[27] Fischer, M.J., Lynch, N.A. and Paterson, M.S. 1985. Impossibility of Distributed Consensus with One Faulty Process. *J. ACM*. 32, 2 (Apr. 1985), 374–382.

[28] Fouquet, F., Daubert, E. and Plouzeau, N. 2012. Dissemination of reconfiguration policies on mesh networks. *Distributed Applications and Interoperable Systems* (2012), 16–30.

[29] Garlan, D., Cheng, S.-W., Huang, A.-C., Schmerl, B. and Steenkiste, P. 2004. Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure. *Computer*. 37, 10 (2004), 46–54.

[30] Georgiadis, I., Magee, J. and Kramer, J. 2002. Self-organising software architectures for distributed systems. *Proceedings of the first workshop on Self-healing systems - WOSS '02* (2002), 33–38.

[31] Gerostathopoulos, I., Bures, T., Hnetynka, P., Hujecek, A., Plasil, F. and Skoda, D. 2015. *Meta-Adaptation Strategies for Adaptation in Cyber-Physical Systems*. Technical Report #D3S-TR-2015-01. Department of Distributed and Dependable Systems, D3S-TR-2015-01.

[32] Goldsby, H.J., Sawyer, P., Bencomo, N., Cheng, B.H.C. and Hughes, D. 2008. Goal-Based Modeling of Dynamically Adaptive System Requirements. *Proc. of the 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2008)* (Mar. 2008), 36–45.

[33] Gupta, R.A. and Chow, M.-Y. 2010. Networked Control System: Overview and Research Trends. *IEEE Transactions on Industrial Electronics*. 57, 7 (Jul. 2010), 2527–2535.

[34] Hall, R., Pauls, K., McCulloch, S. and Savage, D. 2011. *OSGi in Action: Creating Modular Applications in Java*. Manning Publications, Stamford, CT.

[35] Hansen, K.M. 2012. Modeling and Analyzing Architectural Change with Alloy. *SAC '10* (2012), 2257–2264.

[36] Hasan, M.S., Yu, H., Carrington, A. and Yang, T.C. 2009. Co-simulation of wireless networked control systems over mobile ad hoc network using SIMULINK and OPNET. *IET Communications*. 3, 8 (2009), 1297.

[37] Hellerstein, J., Diao, Y., Parekh, S. and Tilbury, D. 2004. *Feedback Control of Computing Systems*.

[38] Hinchey, M., Park, S. and Schmid, K. 2012. Building Dynamic Software Product Lines. *Computer*. 45, 10 (Oct. 2012), 22–26.

[39] Hirsch, D., Kramer, J., Magee, J. and Uchitel, S. 2006. Modes for software architectures. *Proc. of the 3rd European conference on Software Architecture, EWSA '06* (2006), 113–126.

[40] Hoelzl, M., Rauschmayer, A. and Wirsing, M. 2008. Engineering of Software-Intensive Systems: State of the

Art and Research Challenges. *Software-Intensive Systems and New Computing Paradigms*. 1–44.

[41] Höst, M., Regnell, B. and Wohlin, C. 2000. Using Students as Subjects—A Comparative Study of Students and Professionals in Lead-Time Impact Assessment. *Empirical Software Engineering*. 214, (2000), 201–214.

[42] IRM-SA Website: 2015. *http://d3s.mff.cuni.cz/projects/irm-sa*. Accessed: 2015-04-23.

[43] Jadbabaie, A., Lin, J. and Morse, A.S. 2003. Coordination of groups of mobile autonomous agents using nearest neighbor rules. *IEEE Transactions on Automatic Control*. 48, 6 (Jun. 2003), 988–1001.

[44] jDEECo Website: 2015. *https://github.com/d3scomp/JDEECo*. Accessed: 2015-04-23.

[45] Kang, K.C., Jaejoon, L. and Donohoe, P. 2002. Feature-oriented product line engineering. *IEEE Software*. 19, 4 (2002), 58–65.

[46] Kephart, J. and Chess, D. 2003. The Vision of Autonomic Computing. *Computer*. 36, 1 (2003), 41–50.

[47] Keznikl, J., Bures, T., Plasil, F., Gerostathopoulos, I., Hnetynka, P. and Hoch, N. 2013. Design of Ensemble-Based Component Systems by Invariant Refinement. *Proc. of CBSE'13* (Vancouver, Canada, Jun. 2013), 91–100.

[48] Keznikl, J., Bures, T., Plasil, F. and Kit, M. 2012. Towards Dependable Emergent Ensembles of Components: The DEECo Component Model. *Proc. of WICSA/ECSA'12* (Aug. 2012), 249–252.

[49] Kramer, J. and Magee, J. 2007. Self-managed systems: an architectural challenge. *Proc. of FOSE'07* (Minneapolis, USA, May 2007), 259–268.

[50] Lamsweerde, A. Van 2008. Requirements engineering: from craft to discipline. *16th ACM Sigsoft Intl. Symposium on the Foundations of Software Engineering* (Atlanta, USA, Nov. 2008), 238–249.

[51] Lamsweerde, A. Van 2009. *Requirements Engineering: From System Goals to UML Models to Software Specifications*. John Wiley and Sons.

[52] Lamsweerde, A. Van and Letier, E. 2000. Handling obstacles in goal-oriented requirements engineering. *IEEE Transactions on Software Engineering*. 26, 10 (2000), 978–1005.

[53] LeBerre, D. and Parrain, A. 2010. The Sat4j Library, release 2.2. *Journal on Satisfiability, Boolean Modeling and Computation*. 7, (2010), 59–64.

[54] McKinley, P.K., Sadjadi, S.M., Kasten, E.P. and Cheng, B.H.C. 2004. Composing adaptive software. *Computer*. 37, 7 (2004), 56–64.

[55] Morandini, M., Penserini, L. and Perini, A. 2008. Automated Mapping from Goal Models to Self-Adaptive Systems. *2008 23rd IEEE/ACM International Conference on Automated Software Engineering* (Sep. 2008), 485–486.

[56] Morandini, M. and Perini, A. 2008. Towards Goal-Oriented Development of Self-Adaptive Systems. *Proceedings of the 2008 International Workshop on Software Engineering for Adaptive and Self-managing Systems - SEAMS '08* (Leipzig, Germany, May 2008), 9–16.

[57] Morin, B., Barais, O., Jezequel, J.-M., Fleurey, F. and Solberg, A. 2009. Models at Runtime to Support Dynamic Adaptation. *Computer*. 42, 10 (2009), 44–51.

[58] Morin, B., Barais, O., Nain, G. and Jezequel, J. 2009. Taming Dynamically Adaptive Systems Using Models and Aspects. *Proc. of the 31st International Conference in Software Engineering, ICSE '09* (2009), 122–132.

[59] Murguzur, A., Capilla, R., Trujillo, S., Ortiz, Ó. and Lopez-Herrejon, R.E. 2014. Context Variability Modeling for Runtime Configuration of Service-based Dynamic Software Product Lines. *Proceedings of the 18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools - Volume 2* (New York, NY, USA, 2014), 2–9.

[60] Murray, R.M., Astrom, K.J., Boyd, S.P., Brockett, R.W. and Stein, G. 2003. Future Directions in Control in an Information-Rich World. *Control Systems, IEEE*. 23, 2 (2003), 1–21.

[61] Mustafiz, S., Denil, J., Lúcio, L. and Vangheluwe, H. 2012. The FTG+ PM framework for multi-paradigm modelling: An automotive case study. *Proceedings of the 6th International Workshop on Multi-Paradigm Modeling* (2012), 13–18.

[62] Olfati-Saber, R., Fax, J.A. and Murray, R.M. 2007. Consensus and Cooperation in Networked Multi-Agent Systems. *Proceedings of the IEEE*. 95, 1 (Jan. 2007), 215–233.

[63] Oreizy, P., Gorlick, M.M., Taylor, R.N., Heimbigner, D., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D.S. and Wolf, A.L. 1999. An Architecture-Based Approach to Self-Adaptive Software. *Intelligent Systems and their Applications, IEEE*. 14, 3 (1999), 54 – 62.

[64] Robinson, W.N. 2005. A requirements monitoring framework for enterprise systems. *Requirements Engineering*. 11, 1 (Nov. 2005), 17–41.

[65] Salehie, M. and Tahvildari, L. 2009. Self-Adaptive Software: Landscape and Research Challenges. *ACM Transactions on Autonomous and Adaptive Systems*. 4, 2, May (2009), 1–40.

[66] Sawyer, P., Bencomo, N., Whittle, J., Letier, E. and Finkelstein, A. 2010. Requirements-Aware Systems: A Research Agenda for RE for Self-adaptive Systems. *Proc. of the 18th IEEE International Requirements Engineering Conference* (Sep. 2010), 95–103.

[67] Sawyer, P., Rocquencourt, I., Mazo, R., Diaz, D., Salinesi, C. and Paris, U. 2012. Using Constraint Programming to Manage Configurations in Self-Adaptive Systems. *Computer*. 45, 10 (2012), 56–63.

[68] Sentilles, S., Vulgarakis, A., Bures, T., Carlson, J. and Crnkovic, I. 2008. A component model for control-intensive distributed embedded systems. *Proc. of the 11th International Symposium on Component-Based Software Engineering* (Oct. 2008), 310–317.

[69] Seuken, S. and Zilberstein, S. 2012. Improved Memory-Bounded Dynamic Programming for Decentralized POMDPs. *Proc. of the Twenty-Third Conference on Uncertainty in Artificial Intelligence* (2012).

[70] Sha, L. and Meseguer, J. 2008. Design of Complex Cyber Physical Systems with Formalized Architectural Patterns. *Software-Intensive Systems and New Computing Paradigms*. M. Wirsing, J.-P. Banâtre, M. Hölzl, and A. Rauschmayer, eds. Springer. 92–100.

[71] Sheskin, D.J. 2011. *Handbook of Parametric and Nonparametric Statistical Procedures*. Chapman and Hall/CRC.

[72] Shoham, Y. and Leyton-Brown, K. 2008. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press.

[73] Souza, V.E.S., Lapouchnian, A., Robinson, W.N. and Mylopoulos, J. 2013. Awareness Requirements. *Software Engineering for Self-Adaptive Systems II*. Springer Berlin Heidelberg. 133–161.

[74] Spaan, M.T.J., Oliehoek, F.A. and Vlassis, N. 2008. Multiagent Planning under Uncertainty with Stochastic Communication Delays. *Proc. of Int. Conf. on Automated Planning and Scheduling* (2008), 338–345.

[75] The Orocos Real-Time Toolkit: *http://www.orocos.org/wiki/orocos/rtt-wiki*.

[76] Valtazanos, A. and Steedman, M. 2014. Improving Uncoordinated Collaboration in Partially Observable Domains with Imperfect Simultaneous Action Communication. *Proc. of the Workshop on Distributed and Multi-Agent Planning in ICAPS* (2014), 45–54.

[77] Whittle, J., Sawyer, P. and Bencomo, N. 2010. RELAX: A Language to Address Uncertainty in Self-Adaptive Systems Requirements. *Requirements Engineering*. 15, 2 (2010), 177–196.

[78] Wohlin, C., Runeson, P., Host, M., Ohlsson, M.C., Regnell, B. and Wesslen, A. 2012. *Experimentation in Software Engineering*. Springer.

[79] Wu, F., Zilberstein, S. and Chen, X. 2011. Online Planning for Ad Hoc Autonomous Agent Teams. *Proc. of the 22nd International Joint Conference on Artificial Intelligence* (2011), 439–445.

[80] Wu, F., Zilberstein, S. and Chen, X. 2011. Online planning for multi-agent systems with bounded communication. *Artificial Intelligence*. 175, 2 (Feb. 2011), 487–511.