Adaptation in Cyber-Physical Systems: from System Goals to Architecture Configurations

Tomas Bures\textsuperscript{1}, Ilias Gerostathopoulos\textsuperscript{1}, Petr Hnetynka\textsuperscript{1}, Jaroslav Keznikl\textsuperscript{1}, Michal Kit\textsuperscript{1}, Frantisek Plasil\textsuperscript{1}, Noël Plouzeau\textsuperscript{2}

\textsuperscript{1}Charles University in Prague
Faculty of Mathematics and Physics
Prague, Czech Republic

\textsuperscript{2}IRISA
University of Rennes 1
Rennes, France

Abstract: Design of self-adaptive Cyber-Physical Systems (CPS) operating in not fully anticipated environments is a significant challenge, especially if the design is to provide for a sufficient level of dependability. This stems partly from the fact that the concerns of self-adaptivity and dependability are to certain extent contradictory. In this paper, we present an extension to IRM (Invariant Refinement Method) – a design method and associated formally grounded model targeting CPS – that addresses self-adaptivity while preserving the dependability aspects. Specifically, we extend IRM to provide traceability between system requirements, distinct situations in the environment, and predefined configurations of system architecture. Additionally, based on this traceability, we propose a method for adaptation at runtime that allows coping with unanticipated situations. As a proof of concept, we implemented the proposed method for the DEECo component model, based on dynamic ensembles of components.

Version: March 18th, 2014

This work was partially supported by the EU project ASCENS 257414. 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.
Adaptation in Cyber-Physical Systems: from System Goals to Architecture Configurations

Tomas Bures\textsuperscript{1,2} \texttt{bures@d3s.mff.cuni.cz} \hspace{1cm} Ilias Gerostathopoulos\textsuperscript{1} \texttt{iliasg@d3s.mff.cuni.cz} \hspace{1cm} Petr Hnetynka\textsuperscript{1} \texttt{hnetynka@d3s.mff.cuni.cz} \hspace{1cm} Jaroslav Keznikl\textsuperscript{1,2} \texttt{keznikl@d3s.mff.cuni.cz}

Michal Kit\textsuperscript{1} \texttt{kit@d3s.mff.cuni.cz} \hspace{1cm} Frantisek Plasil\textsuperscript{1} \texttt{plasil@d3s.mff.cuni.cz} \hspace{1cm} Noël Plouzeau\textsuperscript{3} \texttt{noel.plouzeau@irisa.fr}

\textsuperscript{1}Charles University in Prague 
Faculty of Mathematics and Physics 
Prague, Czech Republic 
\textsuperscript{2}Institute of Computer Science 
Academy of Sciences of the Czech Republic 
Prague, Czech Republic 
\textsuperscript{3}IRISA 
University of Rennes 1 
Rennes, France

ABSTRACT

Design of self-adaptive Cyber-Physical Systems (CPS) operating in not fully anticipated environments is a significant challenge, especially if the design is to provide for a sufficient level of dependability. This stems partly from the fact that the concerns of self-adaptivity and dependability are to certain extent contradictory. In this paper, we present an extension to IRM (Invariant Refinement Method) – a design method and associated formally grounded model targeting CPS – that addresses self-adaptivity while preserving the dependability aspects. Specifically, we extend IRM to provide traceability between system requirements, distinct situations in the environment, and predefined configurations of system architecture. Additionally, based on this traceability, we propose a method for adaptation at runtime that allows coping with unanticipated situations. As a proof of concept, we implemented the proposed method for the DEECo component model, based on dynamic ensembles of components.

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). Examples of CPS are numerous: systems for intelligent car navigation, smart electric grids, emergency coordination, to name just a few.

Design of such systems is a challenging task, as one has to deal with the different, and to an extent contradictory, concerns of open-endedness, dependability and self-adaptivity. As modern CPS are ubiquitous ecosystems, they need to be open-ended, i.e., able to incorporate new subsystems on demand. Since they often host safety-critical applications, they have to feature dependable (mostly safe and predictable) behavior, even in presence of high dynamism. Since they operate in disparate environments corresponding to the ever-changing physical world, they need be self-adaptive \cite{31} and self-calibrating. An additional issue emerges when these environments are not fully anticipated at design time; this calls for methods that deal with the uncertainty present in the operational environments of CPS.

Existing methods largely fail to address this challenging synergy of dependability and self-adaptivity in presence of environment uncertainty. They typically address only one side of the problem – e.g., open-endedness and conceptual autonomy are tackled by agent-oriented methodologies \cite{11,33}; dependability assurances with very limited self-adaptivity is brought by component-based mode-switching methods \cite{15,20}. Environment uncertainty is also often viewed as a separate problem \cite{8}. What is missing is a design method and associated model that would specifically target the development of open-ended dependable and self-adaptive CPS and allow for management of dependability w.r.t. not fully anticipated environments both at design time and runtime.

Delving into the concrete issues, self-adaptive CPS need to be able to adapt to distinct realizations of system requirements in distinct runtime situations (i.e., states of the environment as observed by the system). Adopting an architectural view, these realizations take the form of distinct architecture configurations, which provide the basis for architectural adaptation at runtime. Nevertheless, in order to ensure dependability, these configurations need to be also accounted for at design time.

Advantageously, being reflected in system requirements, the architecture 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 \cite{17}.

A challenge though is that due to the not fully anticipated environment, the adaptation cannot be fully determined at design time. This calls for a solution, which in anticipated situations would provide fully dependable behavior and in an unanticipated situation would still be able to deal with the situation in a best-effort manner while guaranteeing at least basic safety.

In this paper we provide such a solution by carefully combining design-time requirements elaboration and definition of architecture configurations with runtime-based reasoning so as to gracefully cope with unanticipated situations.

Specifically, we propose an extension to IRM \cite{25} – a design method and associated formally grounded model targeting CPS requirements – to support designing for adaptation, while accounting for dependability. We call this IRM for Self-Adaptivity (IRM-SA). In particular, we extend IRM to provide traceability between system requirements, distinct situations in the environment, and predefined configurations of system architecture. Our detailed research goals are:

\begin{itemize}
  \item to model the design alternatives in the architecture pertaining to distinct situations via systematic, formally-grounded elaboration of system requirements;
\end{itemize}
• to provide an automated technique for obtaining the appropriate architecture configurations based on the modeled design alternatives and the perceived situation, relying on formal analysis of the model;
• to provide strategies to deal with situations that are not anticipated at design time, acknowledging the uncertainty inherent to CPS.

To evaluate the feasibility and practicality of our design method, we have applied it to a firefighter coordination case study – Firefighter Tactical Decision System (FTDS) – developed within project DAUM1. We have also implemented support for runtime adaptation based on IRM-SA in the context of DEECo [5] – a component model facilitating open-ended and highly dynamic CPS architectures.

The paper is structured as follows. Section 2 describes the case study, which is employed throughout the paper, 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 selection of suitable alternatives. Section 7 shows how our method deals with unanticipated situations. Section 8 describes our realization of IRM-SA in the DEECo component system, which we used to technically evaluate our approach. Section 9 provides a discussion that identifies the limits of our approach and outlines promising extensions. Finally, Section 10 discusses the related work, while Section 11 concludes the paper.

2. CASE STUDY

We focus on a simple scenario of the FTDS case study where the firefighters belonging to a tactical group communicate with their group leader who aggregates the information about each group member’s condition and his/her environment (parameters considered are 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 measures 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 detached from the network and (b) autonomy so that they can operate optimally without supervision, while still respecting 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 obsolescence 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).

Another source of problems is the presence of unanticipated situations, as, e.g., in the scenario where a firefighter’s GPS breaks at runtime without having been anticipated at design time. In such a case, the system has to switch to the best available solution and still satisfy its goal of monitoring the firefighter’s position in a best effort manner.

3. BACKGROUND

3.1 Invariant Refinement Method

Invariant Refinement Method (IRM) [25] is a goal-oriented design method targeting the domain of CPS. IRM builds on the idea of iterative refinement of requirement specifications yielding low-level obligations which can be operationalized by system agents. Contrary to common goal-oriented modeling approaches (e.g., KAOS [27], Tropos® [4]), IRM incorporates the notion of feedback loops present in autonomic systems. 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. Another key advantage is that there exists a one-to-one mapping from the IRM abstractions to DEECo concepts (Section 3.2), allowing for a seamless transition between the IRM-based specification and component-based architecture and implementation.

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 normality 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 invariant (1): “GM keeps track of the condition of his/her group’s members” (Figure 1).

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., [28],[4]), only software-controlled actors are considered. The two components identified in the running example are GroupMember and GroupLeader.

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

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, Figure 1. 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. \( I_{s1} \land \ldots \land I_{sn} \Rightarrow I_p \) (entailment)
2. \( I_{s1} \land \ldots \land I_{sn} \Rightarrow false \) (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 GroupLeader’s side – invariant (4) – and the necessity to filter the data to identify GroupMembers that are in danger – invariant (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., invariant (5)) and exchange invariants (labeled \( X \), e.g., invariant (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 3.2). Conversely, exchange invariants are maintained by component interaction, typically taking the form of knowledge exchange within a group of components (Section 3.2). 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 [25] – often the case in systems of autonomous agents [43]; inherently, a belief can get outdated and needs to be systematically updated by knowledge exchange in the timeframe decided at the design stage.

### 3.2 DEECo Component System

Dependable Emergent Ensemble of Components (DEECo) [5][24] is component system specifically tailored for building CPS with a high degree of dynamicity in their operation. In DEECo, components are autonomous units of computation and deployment. Each of them comprises knowledge and processes. Knowledge is a hierarchical data structure representing the internal state of the component. A process operates upon the knowledge and features a cyclic execution based on the concept of feedback loop [36], being thus similar to a process in real-time systems. In DEECo, 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 [12][21]. Ensembles are dynamically created based on the state of components and external situation (e.g., when a group of firefighters are physically close together, they form an ensemble). Within an ensemble, the runtime framework of DEECo performs knowledge exchange between the associated components – essentially updating their beliefs.
Therefore, we extend the IRM design model with the mechanism of alternative decomposition – OR-decomposition – along with the concept of situation. An overview of the extended graphical representation for IRM-SA is depicted in Figure 3.

Essentially, OR-decomposition denotes a variation point where each of the children represents a refinement alternative, resulting in a potentially different architecture design (specifying different processes and knowledge exchange). 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 refinement alternative can be further refined via another OR-decomposition. Formally, an OR-decomposition of a parent invariant $I_p$ into the sub-invariants $I_{s1}, ..., I_{sn}$ is a refinement if it holds that:

1. $I_{s1} \lor \cdots \lor I_{sn} \Rightarrow I_p$ (alternative entailment)
2. $I_{s1} \lor \cdots \lor I_{sn} \nRightarrow \text{false}$ (alternative consistency)

Notably, each refinement alternative is specific to a certain situation, i.e., a specific state of the environment. In IRM-SA (as well as in pure IRM), a particular state of the environment is captured via an assumption. Thus, each refinement alternative in OR-decomposition is associated with an assumption, capturing the corresponding situation.

As an example, consider the leftmost part of the refinement of invariant (3): “GL keeps track of the condition of the relevant members” in Figure 2, which captures two variants corresponding to the situations where either some firefighter in the group is in danger or none is. In the former case (left alternative), invariant (8) is also included – expressing the necessity to inform the other firefighters in the group that a member is in danger.

Note that, if an alternative in an OR-decomposition is further refined in terms of an AND-decomposition, we omit the invariant representing the alternative and connect the AND-decomposition directly to the OR-decomposition to improve readability.

We distinguish two kinds of assumptions: computable and non-computable. While a computable assumption can be programatically directly evaluated by monitoring, a non-computable assumption serves primarily for design and review-based validation purposes. Thus, the assumptions associated with refinement alternatives should be preferably computable.

An example of a non-computable assumption is (28): “GM indoors” in Figure 4, the importance of which lies in serving as a guideline in the design of this process. Another example of a non-computable assumption is (20): “No life threat” in Figure 4, giving a high-level description of the situation. It is AND-decomposed into the computable assumptions (22) and (23), representing two orthogonal concerns, which can be evaluated by monitoring the GroupMember’s internal state.

The situations corresponding to refinement alternatives in an OR-decomposition may overlap, signifying that more variants may be applicable in certain situations. Thus, the alternatives can be associated with preferences (Section 6.2). The preferences define a total preorder of alternatives of one OR-decomposition. In diagrams, we capture this by ranks (1, 2, etc.) attached to alternatives. If no rank is given, 1 is assumed.

Moreover, dependencies may exist between (invariants in) alternatives across different OR-decompositions, representing constraints of the physical world. These dependencies are captured in the IRM-SA model by directed links labeled with “requires”, resp. “collides”, which capture the constraint that the source alternative can appear in an architecture configuration only with, resp. without, the target alternative. For example, in order for invariant (33) in Figure 4 to appear in an architecture configuration, invariant (18) 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.

### 5.1 The Modeling Process

The process of systematic identification and modeling of all the possible variation points with the corresponding alternatives and situations during invariant refinement in IRM-SA is closely related to the identification of adaptation scenarios in a Dynamically Adaptive System (DAS) [17]. 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 [29]. Performance and resource optimization concerns can also guide the identification of alternatives.

For example, the rationale behind the OR-decomposition of the left-most part of the AND-decomposition of (11) in Figure 4 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 a critical situation (e.g., a firefighter is not moving, invariant (16)), when the accuracy of external temperature monitoring plays an important role in the assessment of the criticality of the firefighter’s condition.

On the contrary, the OR-decomposition of (27) in Figure 4 has its rationale in a functional constraint: since GPS is usually not available within a building, a firefighter’s position has to be monitored in another way in such a case, e.g., through an indoors tracking system [6]. This is an example of a technology-driven process of identification of alternatives, where the underlying infrastructure significantly influences the possible range of adaptation scenarios [17]. For example, it would be meaningless 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.

The rest of the process, such as deciding when to stop the refinement, remains the same as in the pure IRM [25].

6. SELECTING ARCHITECTURE CONFIGURATIONS BY SAT SOLVING

As outlined in Section 4, given an IRM-SA model, the selection of an architecture configuration for a given situation can be advantageously done by formal analysis of the model. In this section we describe the formal framework we use for the selection. Specifically, we describe the selection as a Weighted Partial MAX-SAT (WPMSAT) problem [2], which is the problem of selecting a satisfying valuation of a propositional formula such that the sum of costs of violated soft clauses is minimized (soft clauses are the clauses of the formula that are permitted to be falsified in a satisfying valuation). Consequently, a conventional solver can be used for exploring the solution space to select the applicable alternatives for the architecture configuration.

As an aside, 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 WPMSAT), but rely on the well-known fact that every propositional formula can be converted to an equivalent set of CNF clauses.

6.1 Configurations

Formally, the problem of selecting an applicable configuration is the problem of constructing a set \( C \) (representing the configuration) 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 \( I_1, ..., I_m \), then \( I_p \in C \) iff all \( I_1, ..., I_m \in C \); (iii) if an invariant \( I_p \) is decomposed by an OR-decomposition to

1. // 1. hard clauses
2. // 1.1 configuration constraints based of the IRM model
3. // top level decomposition in Figure 4
4. \( s_{1,1} \land s_{16} \land s_{27} \Rightarrow s_{f1} \) // \( s_{f1} \) represents the anonymous invariant in the AND decomposition of (11)
5. \( s_{22,1} \lor s_{22,2} \lor s_{20} \Rightarrow s_{j2} \) // \( s_{j2} \) is a copy of \( s_{20} \)
6. // decomposition level 1 in Figure 4
7. \( s_{21,1} \lor s_{21,2} \lor s_{23} \Rightarrow s_{f1,1} \)
8. \( s_{27,1} \lor s_{27,2} \Rightarrow s_{j2} \)
9. \( s_{23} \land s_{13} \Rightarrow s_{j2,2} \) // \( s_{j2,2} \) is a copy of \( s_{13} \)
10. // decomposition level 2 in Figure 4
11. \( s_{31} \land s_{18} \Rightarrow s_{14,1} \)
12. // decomposition level 3 in Figure 4
13. \( s_{31} \land s_{18} \Rightarrow s_{14,1} \)
14. \( s_{32} \land s_{14} \Rightarrow s_{14,2} \)
15. \( s_{24} \land s_{25} \Rightarrow s_{20} \)
16. \( s_{20} \Rightarrow s_{27,1} \) // non-computable assumption (28) is omitted
17. \( s_{21} \Rightarrow s_{27,2} \) // non-computable assumption (30) is omitted
18. // decomposition level 4 in Figure 4
19. \( s_{15} \land s_{13} \Rightarrow s_{14} \)
20. // decomposition level 5 in Figure 4
21. \( s_{16} \land s_{14} \Rightarrow s_{14} \)
22. // 1.2 only applicable invariants may be selected into a configuration
23. \( s_{15} \Rightarrow s_{14,1} \lor ... \land s_{15} \Rightarrow s_{14,2} \land ... \)
24. // 1.3 determining acceptability according to monitoring
25. // (current configuration as shown in Figure 5)
26. // 1.3.1 active monitoring
27. \( a_{i1} = \ldots \) // true or false based on the monitoring of (11)
28. ... // repeat for \( a_{i1}, a_{i2}, a_{i1, a_{i2}}, a_{i2, a_{i1}}, a_{i2}, a_{i2}, a_{i2} \)
29. ... // repeat for the rest
30. // 2. soft clauses
31. \( s_{11,1,1} = true \) // cost 3 (1*3)
32. \( s_{11,1,2} = true \) // cost 9 (3*3)
33. \( s_{11,1,3} = false \) // cost 6 (2*3)
34. \( s_{21,1} = true \) // cost 1 (1*1)
35. ... // repeat for the other alternatives

Figure 6: Encoding of Figure 4 into WPMSAT.

\( I_1, ..., I_m \), then \( I_p \in C \) iff exactly one of \( I_1, ..., I_m \) is in \( C \). The rules ensure that the configuration \( C \) is well-formed with respect to decomposition. Figure 5 shows a sample configuration (selected invariants are outlined in bold).

Technically, for the sake of encoding the configuration selection as a WPMSAT 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 \( C \) we are looking for by introducing Boolean variables \( s_1, ..., s_n \) such that \( s_i = true \) iff \( I_i \in C \). To ensure \( C \) is well-formed, we introduce clauses over \( s_1, ..., s_n \) reflecting the rules (i)-(iii) above. For instance, the IRM-SA model from Figure 4 will be encoded as shown in Figure 6, lines 4-27.

To ensure that \( C \) is actually applicable w.r.t. the given situation, we introduce Boolean variables \( a_1, ..., a_n \) and add a clause \( s_i \Rightarrow a_i \) for each \( i \in \{1, ..., n\} \) (Figure 6, line 30). 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 \( a_1, ..., a_n \) are bound to reflect the given situation via monitoring the state of the system and environment.
(Figure 6, lines 32-40). How this is exactly done is described in Section 6.3.

Lastly, we include clauses that correspond to dependency relations between invariants as described in Section 5. They take the form of clauses \( s_i = s_j \), resp. \( s_i \Rightarrow \neg s_j \), for any pair of invariants \( l_i, l_j \) that are in a \( s_i \) requires \( s_j \), resp. \( s_i \) collides \( s_j \), relation.

In the resulting WPMSAT instance, the variable \( s_i \) for each top-level invariant \( l_i \) is bound to true to enforce selection of a configuration. A satisfying valuation of such a WPMSAT instance encodes an applicable configuration, while unsatisfiability of the instance indicates nonexistence of an applicable configuration in the current situation.

### 6.2 Preferences among Alternatives

Now we describe how to assign costs to clauses to reflect the preferences associated with alternatives in an IRM-SA model. Preferences are captured as a total preorder of alternatives of an OR-decomposition. In this section, for the sake of brevity, we view the total preorder as a numerical ranking, where \( 1 \) corresponds to the most preferred alternatives, 2 to the second most preferred, etc. (Figure 4). Recall that the default rank is 1.

The main idea is that the preferences loose significance by an order of magnitude from top to bottom such that preferences of alternatives that are lower in the IRM-SA model cannot impact the selection of an alternative, which is above them on a path to a top-level invariant. To reflect this in WPMSAT, we have to translate preferences to costs associated with child invariants of each OR-decomposition. The costs have to be set up in such a way that minimizing their sum yields the best configuration with respect to the preorder of preferences.

Technically, we associate every child invariant \( l_i \) of an OR-decomposition with its OR-layer number \( d_i \), which expresses that \( l_i \) is an alternative of \( d_i \)-th OR-decomposition on a path from a top-level invariant (level 1) to a leaf.

Further, for each OR-layer in a IRM-SA model, we define its cost base \( b_l \) in the following way: (a) the lowest OR-layer (i.e., the one with the highest layer number) has cost base equal to 1, (b) the \( j \)th OR-layer has its cost base \( b_j = b_{j+1} \cdot (n_{j+1} + 1) \), where \( n_{j+1} \) denotes the number of all alternatives at the layer \( j + 1 \) (i.e., considering all OR-decompositions at this layer).

With these definitions in place, we define the cost of a child invariant \( l_i \) of a \( d_i \)-th OR-decomposition as a cost \( b_{d_i} \) as \( \text{rank} \cdot b_{d_i} \), where \( \text{rank} \) denotes the rank of the alternative that the invariant \( l_i \) corresponds to. Since the standard formulation of WPMSAT allows associating costs only with clauses, we add a soft-clause \( s_i = \text{true} \) with the cost \( \text{rank} \cdot b_{d_i} \) for each such an invariant \( l_i \).

The structure of our WPMSAT problem along with cost assignment is illustrated in Figure 6, lines 43-46, which shows it for the IRM-SA model in Figure 4. In this example, although Figure 4 is a continuation of Figure 2, we assume for brevity the invariant (11) is at the top level (i.e., we only consider Figure 4).

### 6.3 Determining Acceptability

Determining acceptability of an invariant \( l_i \) (i.e., determining the value of \( a_i \)) is an essential step in IRM-SA. In principle, a valuation of \( a_i \) reflects whether \( l_i \) is applicable w.r.t. the current state of the system and the current situation of the environment. Essentially, \( a_i = \text{false} \) implies that any configuration containing \( l_i \) in its AND-decomposition can be ignored in the configuration selection process.

We determine the value of \( a_i \) via one of the following actions.

1. **Active monitoring.** If \( l_i \) belongs to the current configuration, or is an assumption, and is in such a form that it can be programmatically evaluated, we determine \( a_i \) by evaluating \( l_i \) w.r.t. the current knowledge of the components taking a role in \( l_i \).
2. **Predictive monitoring.** If \( l_i \) is not an assumption and does not belong to the current configuration but is in such a form that it can be programmatically evaluated, we estimate whether \( l_i \) would be satisfied in another configuration if that were chosen.
3. **Implicit determination.** If \( l_i \) is in such a form that it cannot be programmatically evaluated, we assume that it is satisfied. This corresponds to disregarding the invariant from the WPMSAT solving process as the invariant cannot be programmatically evaluated and presumably served only for documentation purposes.

Furthermore, we use two major estimation approaches for predictive monitoring: (a) the developer provides a predicate that assesses whether the invariant would be satisfied based on the data available, and (b) we keep a history of the invariant evaluation.

Naturally, (a) is a preferred option because of higher accuracy. It is especially useful for process invariants, where the developer-supplied predicate may assess not only the validity of process invariant, but also whether the process would be able to perform its computation at all. This can be illustrated on process invariant (31), where the process maintaining it can successfully complete (and thus satisfy the invariant) only if GPS is operational and at least four GPS satellites are visible.

If the developer does not provide an additional predicate for each invariant, we can employ option (b), the goal of which is to prevent oscillations in switching of configurations by remembering that active monitoring found an invariant unacceptable in recent past.

### 7. STRATEGIES FOR DEALING WITH UNANTICIPATED SITUATIONS

Although IRM-SA modeling and reasoning as described in the previous sections relies on anticipated situations, we realize that modern CPS often need to operate in situations that reside outside of their ”envelope of adaptability” [3]. In this section, we explain how IRM-SA tackles this problem at runtime and by re-design. The driving idea is to control the decline of dependability in the system, caused by unanticipated situations, so that the system’s operation degrades gradually in a controlled manner. While in this section we discuss the solution that we have successfully applied to the firefighter case study, we extend the discussion of tackling the problem in Section 9.2.

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 the information about their position as captured by their GPS devices. A problem arises when GPS monitoring breaks (battery drainage or even physical damage of the device): the system would no longer be able to adapt, since this failure was not anticipated at design time. Below, we explain which strategies we propose to cope with such unanticipated situations in IRM-SA.
7.1 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 by providing a number of alternatives to be selected in a particular situation. When the system fails due to an unanticipated situation (hidden assumption), there is a significant chance, that another alternative may be selected to preserve the top-level invariants and be unaffected by this hidden assumption. This makes the overlapping alternatives a natural way of supporting fault-tolerance in the design of the system.

A special case of providing a number of applicable alternatives is that an IRM-SA model typically contains one or more alternatives that have very weak assumptions and minimum preference, so that such alternatives are only chosen as the last option. Such alternatives reflect “fail-safe” modes as typically found in the design of safety-critical systems.

An important aspect of this strategy is that the system in question is not adapted (reconfigured) just by chance, but in a way based solely on the designed IRM-SA model. This reduces the emergent behavior and allows sustaining an adequate level of dependability. Further, the adaptation is based on monitoring of leaf invariants as well as higher-level invariants. This allows observing failures of the system (typically stemming from flawed design and/or hidden assumptions) when either (i) a process invariant fails, or (ii) a higher-level invariant is not satisfied, although its descendant process invariants hold and the processes maintaining them work flawlessly. The bottom line is that by adapting to another alternative, the system can find a successful way to address an unanticipated situation.

In the running example, this strategy is employed in two OR-decompositions in Figure 4. Here, the left-most part of the decomposition of invariant (11) “GM::sensorData is determined” is meant to be maintained differently when the associated group member is in critical situation (left alternative), when a nearby group member is in danger (right alternative), and when no life is in threat (middle alternative). The situation, corresponding to the middle alternative, stands as a counterpart of the other two situations, which are not mutually exclusive. The invariant (27) “GM::position is determined” is meant to be maintained in the two situations represented by the assumptions (28) “GM indoors” and (30) “GM outdoors”. These 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 indoor tracking system to track his position; this alternative is automatically chosen when the GPS unexpectedly malfunctions.

7.2 Re-design Strategy
The re-design strategy is applied in design evolution process driven by the designer – he/she analyzes occurrences of unanticipated situations that the system had to cope with by adaptation (via employing the runtime strategy). The designer reviews the solutions taken by the system and revises the design of IRM-SA model accordingly. Such a revision can range from inclusion of a single invariant to restructuring of the whole IRM-SA model.

An important aid in such a revision is the fact that, in each invariant refinement, there exists the implication relationship between the sub-invariants and the parent invariant. By monitoring the satisfaction of the parent invariant $I_p$ and sub-invariants $I_{1}, ..., I_{n}$, it is possible to narrow down the adaptation problem and infer a suitable way of addressing it. In particular, an adaptation problem occurs when:

(a) $I_p$ is AND-decomposed, all non-process invariants among $I_1, ..., I_n$ hold but $I_p$ does not hold. This points to a hidden 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.

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” – process invariant (31) in Figure 4 – and its parent “GM::position is determined” – invariant (27) – do not hold, which is a symptom for an unanticipated situation. Indeed, the root cause is that GPS was considered responsive at all times. To mitigate this problem, we employ the evolution of the running example as presented in Figure 7. There, the unanticipated situation has become explicit and is used to drive the adaptation. Specifically, in this new situation, the system still satisfies the invariant (27) “GM::position is determined” by switching to the right-most alternative. In such a case, the GroupMember’s position is determined by aggregating the positions of the nearby firefighters – invariant (37) – and estimating its own position based on these positions and the radius of search, through determining the maximum overlapping area – invariant (38).

8. REALIZATION IN DEECo
To evaluate our approach and prove its feasibility, we have implemented² IRM-SA-based adaptation in the jDEECo framework [10], which is the Java realization of the DEECo component model [5]. jDEECo provides an internal Java DSL for developing DEECo components and ensembles an allows their distributed execution.

---

² Available at [http://d3s.mff.cuni.cz/projects/irm-sa](http://d3s.mff.cuni.cz/projects/irm-sa)
For integration of IRM-SA-based adaptation, we employ a models-at-runtime approach [34]. In particular, we maintain an in-memory representation of the IRM-SA model, as well as explicit association of runtime concepts (i.e., components, component processes, and ensembles) with the corresponding IRM-SA invariants. This allows for traceability between requirements, design, and implementation, which is essential for the IRM-SA-based adaptation (especially for the evolutionary aspects discussed in Section 7.2).

The central part of the runtime architecture of the adaptation layer (Figure 8) is the central Adaptation Manager (AM), which aggregates component-local monitoring results, ensures monitoring of invariants distributed across multiple components, performs the selection of an architecture configuration according to Section 6, and triggers adaptation if necessary. Internally, AM employs SAT4J solver [41], mainly due to its seamless integration into Java and good support for optimization variants of SAT. Also, it is able to easily cope with the sizes of the WPMSAT instances in our experiments.

For distributed communication between AM and schedulers/monitors, we internally use the jDEECo distributed knowledge management mechanism. The adaptation itself takes the form of conditional scheduling of component processes and ensemble knowledge exchange. Specifically, as shown in Figure 8, the standard jDEECo scheduler in each instance of the jDEECo runtime framework is instructed by AM to start/stop scheduling processes and knowledge exchange according to the selected architecture configuration.

8.1 Monitoring
To implement monitoring, we have provided an internal Java DSL for definition of monitors. As described in Section 6.3, there are two types of invariant monitoring: active and predictive. While the former checks whether or not a currently active invariant is maintained (e.g., whether a process is executing correctly), the latter is responsible for assessing if an idle invariant can be potentially maintained if selected. Thus far, we have used an implementation of predictive monitoring for both component processes and knowledge exchange based on observing historical results of active monitoring as described in Section 6.3. For assumptions, we have used only active monitoring techniques.

Technically, we have realized a monitor as a method returning a Boolean value, which is associated with an invariant. For instance, a monitor for the invariant (31) “GM::position is determined from GPS” from Figure 4 is illustrated in Figure 9; the monitor checks correct operation of the corresponding process by inspecting its output and checking the status of the GPS device. In our prototype implementation, the execution of monitors is not distributed among the involved components, but is driven directly by AM.

8.2 Role Assignment
An important issue for runtime application of IRM-SA-based adaptation is the task of role assignment. This issue stems from the fact that an IRM-SA model is parametric with respect to roles; i.e., the IRM-SA model does not refer to concrete components but only to their abstract “prototypes”, expressed by their expected knowledge and constrained by their roles in the invariants. Thus, before executing the selection of a configuration, AM has to assign each role to a concrete component instance.

Naturally, there are many ways of assigning roles to components. For instance, consider several group leaders and group members from the case study, where some of the components include both knowledge expected from a leader and a member. Here, AM has to perform architecture-variant selection for each suitable role assignment.

In principle, AM can try all the possible role assignments (w.r.t. expected knowledge), and the assumptions in the IRM-SA model will filter out the irrelevant ones. To make this process more effective at runtime, we further pre-filter the structurally possible assignments. Specifically, we assume that the top-level invariant(s) of a given IRM-SA model are accompanied with a top-level assumption, such as (2) in Figure 2. We then use this assumption to filter-out irrelevant role assignments. As an aside, for roles with multiple cardinality we always assign the biggest possible (in inclusion) set of components.

9. DISCUSSION
In this section, we evaluate the benefits and potential limitations of our approach and outline prospective extensions and research challenges.

9.1 Decentralizing Configuration Selection
To retain a sufficient level of component autonomy and overall system efficiency, IRM-SA requires potentially distributed and decentralized decision process. Currently, our jDEECo realization of IRM-SA (Section 8) relies on a single centralized Adaptation Manager (AM). This creates a performance bottleneck and a single point of failure, as it prevents components to act...
autonomously after communication with AM was severed. Below, we discuss two major solutions to address this issue.

The first solution is based on the autonomy of components in DEECo. Here components inherently maintain a belief about the knowledge of other relevant components (as a consequence of performing knowledge exchange in a distributed manner). We take an advantage of this by associating each invariant with a component that is responsible for its periodic monitoring and by including results of the monitoring in the component’s knowledge.

A natural component candidate to be associated with an invariant is the component knowledge of which the invariant refers to. If several such components exist, we select such a component that by its design aggregates most of the knowledge of other candidate components (e.g., the Group Leader component aggregates most of the knowledge of the Group Member components).

By including the monitoring results in a component’s knowledge, other components may spread them via knowledge exchange and beliefs across the system. This allows each component to perform the configuration selection described in Section 6 locally.

A problem here is that a belief is inherently outdated, and thus the outcome of selection may differ in several components. To address this problem, each component performs the global configuration selection locally and announces the outcome to all of the other components. When the outcome is the same for all components, it is accepted and applied. If there are any conflicts, the components resolve them by updating their beliefs and trying again. Although this may result in a conflict resolution phase that never ends (since knowledge is updated asynchronously), it is assumed that the situations that trigger adaptation change less frequently so that conflicts will be resolved eventually.

This solution is robust w.r.t. disconnections, since in the case a node gets disconnected; it will keep performing the adaptation autonomously, based on its latest belief, until the belief is updated.

As a more elaborate solution, we foresee usage of methods based on the distributed constraint optimization algorithms (COP), e.g., Asynchronous Backtracking [46], where each component would be responsible for assigning the $a_i$ and $s_i$ variables in IRM-SA. The responsibility for assigning these variables will be either distributed randomly among the components involved in the corresponding invariants, or centralized in a randomly selected component. Specifically, the use of distributed SAT/COP algorithms [30][39] is promising. The challenge with distributed constraint optimization algorithms is how to achieve the operation in presence of disconnections, since these algorithms invariably require strong connectivity of all nodes.

9.2 Relaxing Requirements at Runtime via Fuzzy Logic

Our approach reaches its limits when there is no configuration that fully satisfies the top-level invariants of an IRM-SA model. In such a case the WPM SAT procedure (Section 6), does not provide any solution. A way of addressing this issue is to “relax” system requirements by finding an architectural configuration that satisfies the top-level invariants “to the best extent”. This can be achieved by employing a monitoring that provides the level of invariant acceptability $a_i$ as a continuous value in the interval $[0,1]$ (as opposed to true/false in bi-state logic).

A challenge here is how to define the aggregated level of acceptability for a whole configuration. This involves reflecting preferences of alternatives, acceptability of invariants, and acceptability of the refinement relationship in both AND- and OR-decompositions.

More precisely, if an invariant $I_p$ is AND-decomposed to $I_{1}, ..., I_{m}$, then the aggregate acceptability of $I_p$ should be a weighted combination of (a) acceptability of $I_p$ (as established by monitoring), and (b) aggregated acceptability of $I_i$ for $i = 1 ... m$. The rationale behind this is that the system should not satisfy the higher-level invariant $I_p$ by “accident” but rather by “conscious effort”, which in terms of invariant acceptability means that the invariants $I_{1}, ..., I_{m}$ should be also satisfied to an adequate level.

In this respect, the t-norm fuzzy logic [18] (e.g., Product fuzzy logic) provide a sound semantics for aggregating acceptability levels, especially when combined with probability weighting [1] to capture the fact that the acceptability of $I_p$ has different significance than aggregate acceptability of $I_i$ for $i = 1 ... m$.

Specifically, when the Product fuzzy logic is employed and the probability weighting function takes form $w(p) = p^a$ (where $p \in [0,1]$ is an acceptability level and $a$ is a constant that defines the concavity/convexity of the function), the function for aggregate acceptability of $I_p$ can be computed as $A_p = (a_p)^{w(p)} \prod_{i=1..m}(A_i)^{w_i}$, where $A_i$ denotes the aggregate acceptability of $I_i$, and $a_p$ denotes the level of acceptability of $I_p$ established by monitoring. When the function $A_p$ is applied to the top-level invariants of a configuration $C$, it computes the overall level of acceptability of $C$ and thus it serves as a metric for comparing different configurations. Another important fact is that the function can be simplified to a product of the form $A_p = \prod_{i=1..n}(a_i)^{\beta_i}$, where $i = 1 ... n$ goes recursively over all child invariants of $I_p$ (including $I_p$). Consequently, by applying logarithm-transformation (which transforms the product form to a sum), we can associate inclusion of an invariant $I_i$ in a configuration $C$ with the reward equal to $\log((a_i)^{\beta_i})$. (Technically, this is done by introducing a clause $s_i = false$ with cost equal to the negation of the reward.) This transforms the problem to the WPM SAT problem having the same structure as described in Section 6.1.

10. RELATED WORK

Recently, there has been a growing interest in software engineering research for software-intensive systems with self-adaptive and autonomous capabilities [40]. The comparison of our work with relevant approaches pertains to three essential views of self-adaptation [7], namely requirements, assurances and engineering.

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” [44], 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 [4] produced at design time, and monitored at runtime by the requirements monitoring framework ReqMon [38]. The idea is to have a highly sophisticated logging framework, on top of which a full MAPE-K feedback loop [23] 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 [32][33]. 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 CPS.

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 [13][14]. 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 CPS environments.

For dealing with uncertainty in the requirements of self-adaptive systems, the RELAX language has been proposed [45]. 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 [8], RELAX specifications are integrated into KAOS goal models. Threat modeling à la KAOS [29] is employed to systematically explore (and subsequently mitigate) uncertainty factors that may impact the requirements of a DAS. Compared to IRM-SA, RELAX focuses on the requirements domain and does not progress from goals and requirements to design and implementation.

At the system design and implementation phases, the component-based architectural approach towards self-adaptivity is favored in several works [15][37]. Here, mode switching stands as a widely accepted enabling mechanism, introduced by Kramer and Magee [20][26]. 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 [35], 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 [42].

Similar to IRM-SA, compositional definition of architecture configurations based on elaboration of individual variation points is addressed by feature models [22], employed for modeling Software Product Lines (SPLs) [9]. IRM-SA models not only the individual design alternatives and their relations, as in traditional feature models for SPLs, but also the situations in which these alternatives are to be employed. Compared to design-time architecture variability in SPLs capturing a whole product family, IRM-SA captures architecture variants of a single product and employs them for runtime self-adaptation based on the perceived situation in the environment.

Finally, architecture adaptation based on various constraint solving techniques is not a new idea. A common conceptualization, e.g., in [16][19], 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.

11. CONCLUSION

In this paper, we have presented the IRM-SA method – an extension to IRM that allows designing of open-ended self-adaptive Cyber-Physical Systems (CPS) 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. The key benefits of the proposed IRM-SA method include:

- Explicit traceability between high-level goals, assumptions about the environment, and available architecture configurations. This makes it possible to detect and react to runtime problems caused by flawed design and unanticipated situations.
- Seamless mapping of high-level goals expressed as invariants to the architecture-level constructs of components and ensembles (as provided by DEECo).
- Design of CPS based on the inbuilt notion of feedback loops, which is one of the key abstractions of CPS.

As a proof of concept, we have implemented IRM-SA within jDEECo (a Java realization of the DEECo component model) to prove the feasibility of our approach. We have successfully employed IRM-SA for design of a real-life case study developed in cooperation with professional firefighters.

The specific problems of gradual relaxation of requirements and decentralized autonomous selection of a configuration are in our view very relevant and important, thus stand high in our research agenda. Other items of current and future work include providing a model-based design tool and using the tool to perform empirical evaluation of our approach at a large-scale.

12. REFERENCES


