Feature-driven Mediator Synthesis: Supporting Collaborative Security in the Internet of Things

Amel Bennaceur, Thein Than Tun, Arosha K. Bandara, Yijun Yu, Bashar Nuseibeh

HAL Id: hal-01366261

https://inria.hal.science/hal-01366261

Submitted on 14 Sep 2016

Feature-driven Mediator Synthesis: Supporting Collaborative Security in the Internet of Things

AMEL BENNACEUR, The Open University
THEIN THAN TUN, The Open University
AROSHA K. BANDARA, The Open University
YIJUN YU, The Open University
BASHAR NUSEIBEH, The Open University and Lero - The Irish Software Research Centre

As the number, complexity, and heterogeneity of connected devices in the Internet of Things (IoT) increase, so does our need to secure these devices, the environment in which they operate, and the assets they manage or control. Collaborative security exploits the capabilities of these connected devices and opportunistically composes them in order to protect assets from potential harm. By dynamically composing these capabilities, collaborative security implements the security controls through which security (and other) requirements are satisfied. However, this dynamic composition is often hampered by the heterogeneity of the devices available in the environment and the diversity of their behaviours. In this paper we present a systematic, tool-supported approach for collaborative security where the analysis of requirements drives the opportunistic composition of capabilities in order to realise the appropriate security control in the operating environment. This opportunistic composition is supported through a combination of feature modelling and mediator synthesis. We use features and transition systems to represent and reason about capabilities and requirements. We formulate the selection of the optimal set of features to implement adequate security control as a multi-objective constrained optimisation problem and use constraint programming to solve it efficiently. The selected features are then used to scope the behaviours of the capabilities and thereby restrict the state space for synthesising the appropriate mediator. The synthesised mediator coordinates the behaviours of the capabilities to satisfy the behaviour specified by the security control. Our approach ensures that the implemented security controls are the optimal ones given the capabilities available in the operating environment. We demonstrate the validity of our approach by implementing a Feature-driven medIation for Collaborative Security (FICS) tool and applying it to a collaborative robots case study.

CCS Concepts:•**Security and privacy** $\rightarrow$ *Security requirements;*•**Human-centered computing** $\rightarrow$ *Ubiquitous and mobile computing;*•**Social and professional topics** $\rightarrow$ *Software selection and adaptation;*

General Terms: Security

Additional Key Words and Phrases: Adaptive security, requirements, mediator synthesis, feature models, collaborative adaptation

ACM Reference Format:
Amel Bennaceur, Thein Than Tun, Arosha K. Bandara, Yijun Yu, and Bashar Nuseibeh, 2016. Feature-oriented Composition of Capabilities to Achieve Collaborative Security. *ACM Trans. Cyber-Phys. Syst.* 1, 1, Article 1 (July 2016), 25 pages.
DOI: 0000001.0000001

1. INTRODUCTION

Many secure systems are designed and developed with pre-determined countermeasures without the possibility to adapt to new resources and devices [Yuan et al. 2014].

This work is supported by SFI grant 13/RC/2094, QNRF NPRP 5-079-1-018, and ERC Advanced Grant no. 291652 (ASAP).
Author's addresses: A. Bennaceur, T. T. Tun, A. K. Bandara, Y. Yu, and B. Nuseibeh, The Open University Computing And Communications Jennie Lee Building Walton Hall Milton Keynes MK7 6AA United Kingdom.

With the prevalence of the Internet of Things (IoT), secure systems are expected to make the best use of the resources and devices in the environment as their availability and capabilities change in order to meet their requirements [Asplund and Nadjm-Tehrani 2016]. However, most existing solutions focus on the security threats associated with the IoT rather than the opportunities brought by the IoT to support security [Sicari et al. 2015]. We believe that the IoT can play an important role in enabling security by offering the infrastructure to connect multiple devices on the fly in order to implement adequate countermeasures [Cerf 2015]. In the context of cyber-physical systems, we particularly focus on physical security, that is the protection of material assets from physical attacks (e.g., theft) [Weingart 2000]. In a previous position paper [Bennaceur et al. 2014], we propose *collaborative security* whereby both the selection and implementation of countermeasures are performed at runtime. In this paper, we propose and elaborate a framework for collaborative security that composes the capabilities of multiple, potentially heterogeneous, devices with variable (configurable) behaviours, in order to satisfy their requirements. Our framework revolves around three concepts: *security controls*, *capabilities*, and *mediators*.

— *Security controls* specify the mechanisms that need to be deployed in order to protect assets from harm, i.e. to satisfy security requirements [Haley et al. 2008].
— *Capabilities* describe the features and behaviours of the devices. Features describe what a device can do in the operating environment while the behaviour describes how it interacts with the environment, including other devices.
— *Mediators* coordinate the behaviours represented by multiple capabilities in order to reach a state where the requirements are satisfied.

Collaborative security leverages the capabilities of the connected IoT devices to implement the adequate security controls. The process of enabling collaborative security spans design time and runtime. At design time, the developer of each individual device defines its capability. The set of requirements and the security controls are specified as well. At runtime, the capabilities of the IoT devices are discovered and composed in order to implement the appropriate security control. In other words, software developers have only to specify the models of the devices and security controls while the framework is responsible for realising the collaborations of IoT devices automatically.

Fig. 1 gives an overview of our framework for collaborative security. The requirements are first analysed in order to determine the specification of the appropriate security control that needs to be implemented. Determining the appropriate security controls often requires trading off security against other requirements such as performance or usability and considering the value of the assets, and potential threats [Salehie et al. 2012]. To implement this security control, the available capabilities are then configured and composed. This process is iterative and is performed in two steps. First, *Feature Selection* computes the optimal set of features to be enabled on a subset of capabilities in order to realise a security control (see Fig. 1-❶).

Feature-driven Mediator Synthesis aims to generate a mediator that coordinates the behaviours associated with the selected subset of capabilities in order to satisfy the behavioural specification of the chosen security control (see Fig. 1-❷). Only the behaviour associated with the selected features are analysed during mediator synthesis. If the mediator synthesis fails then a different set of features must be selected and fed back to the synthesis module. When the mediator synthesis succeeds then the collaboration is realised by enabling the selected features and deploying the synthesised mediator. Hence, while the requirement model can be used at design time to specify security controls and their relationships to those requirements, the realisation of those security controls is performed at runtime by composing the capabilities available in the operating environment.

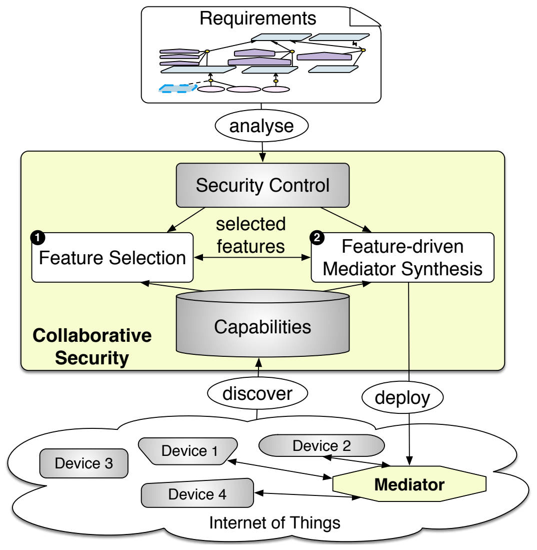


Fig. 1. Overview of our collaborative security framework

This paper focuses on the models and mechanisms to represent, reason about, and mediate capabilities in order to realise security controls. It contributes to three areas:

— *Modelling and reasoning about collaborative security.* There is a conceptual gap between requirements (security and others) and the collaborative behaviour of capabilities necessary to satisfy those requirements. We show that the combination of features and transition systems is a useful abstraction that helps bridge the gap between requirements and capabilities. From a modelling perspective, features give a macro-view of capabilities and security controls as a set of functionalities while transition systems describe how to interact with the capabilities and specify the intended behaviour of security controls. From a reasoning perspective, feature analysis drives the selection of capabilities while behavioural analysis drives the composition of those capabilities to implement a security control.
— *Feature-driven mediator synthesis.* To implement security controls, we compose capabilities considering both their features and behaviours. We first select a set of features to realise a security control. The selected features must also optimise quality attributes such as performance or energy consumption. We formulate the selection of features as a *multi-objective constrained optimisation problem*, which can be efficiently solved using constraint programming [Rossi et al. 2006]. The selection of features allows us to scope the behaviours represented by the capabilities and thereby reduce the analysis space for mediator synthesis. The synthesis algorithm ensures that the composition of the behaviours represented by the capabilities together with the mediator is deadlock-free and reaches a state where the requirements are satisfied. Rather than focusing on a specific algorithm for mediator synthesis, which we tackle elsewhere [Bennaceur and Issarny 2015], we show how these techniques can be improved through feature selection.
— *Tool-support for collaborative security.* We demonstrate the feasibility of our approach by implementing a collaborative security framework, FICS (Feature-driven medIation for Collaborative Security), and evaluate it using a collaborative robotics case study. More specifically, we show using a proof-of-concept demonstrator how two robots—a humanoid robot and a vacuum cleaner—are made to collaborate in

order to implement an additional security control for protecting a mobile phone from theft. The tool and all models discussed in the paper are available at `http://sead1.open.ac.uk/fics/`.

The paper is structured as follows. Section 2 introduces the collaborative robots case study, which we use to illustrate and then evaluate our approach. It also describes our collaborative security framework using Jackson and Zave's framework for requirements engineering [Jackson and Zave 1995]. Section 3 presents the formalism we use to specify our inputs, that is requirements and capabilities. Section 4 details the selection of capabilities. Section 5 moves to the composition of capabilities using mediators. Section 6 evaluates our framework both theoretically and practically. Section 7 examines related work. Finally, Section 8 concludes the paper and discusses future work.

2. OVERVIEW

In this section we introduce our research questions using a collaborative robots example and outline our approach using Jackson and Zave's framework for requirements engineering [Jackson and Zave 1995].

2.1. Motivating Example: Collaborative Robots for Home Security

Traditionally, home security systems require buying and installing multiple cameras as well as a software solution for monitoring and notifications. These systems are rather static and cannot readily adapt to changes in user requirements or to limited resources such as the users renting their house. On the other hand, in 2013, 2.7 million domestic (household) robots (e.g., vacuum and floor cleaning, lawn-mowing robots) and about 1.2 million entertainment robots (e.g., toy robots, hobby systems, and educational robots) were sold [IFR Statistical Department 2014]. Furthermore, Gartner estimates that the typical family home could contain several hundred smart devices by 2022 [van der Meulen and Rivera 2014]. The proliferation of these devices illustrates how the IoT creates new opportunities for protecting our home and the valuable assets therein at a lower cost. In our example, the security requirement is to protect a phone from theft. Note that the focus is on protecting the physical object rather than the data within the phone. In this paper we consider the case of two robots: a programmable autonomous vacuum cleaner (iRobot Create[1]) and a humanoid robot (NAO[2]). Both robots have task-level autonomy [Brooks 2009] in the sense that they are given specific tasks which they decompose and achieve by themselves. For example, we can command NAO to standup, and NAO controls the different joints and motors to perform this task.

Fig. 2 sketches our collaborative robots case study. We use extracts from this case study throughout the paper to illustrate our approach while a demonstration video can be found at `http://sead1.open.ac.uk/fics/`.

Several security controls can be used to protect the phone from theft when the user leaves it unattended: (i) calling out to the user, (ii) hiding the phone in a safe place, or (iii) locking the door of the room in which the phone is. The choice of the appropriate security control may depend on other requirements. In our case study, hiding the phone is preferred to locking the door in order to maximise the accessibility of the room, that is a usability requirement. Although, the Lock can be used to make the room safe and thereby protect the phone, making the two robots, NAO and iRobot Create, collaborate allows us to protect the phone from theft while ensuring that the room remains accessible. In other words, our goal is to leverage the capabilities of NAO

[1]`http://www.irobot.com/us/learn/Educators/Create.aspx`
[2]`http://www.aldebaran-robotics.com/en/`

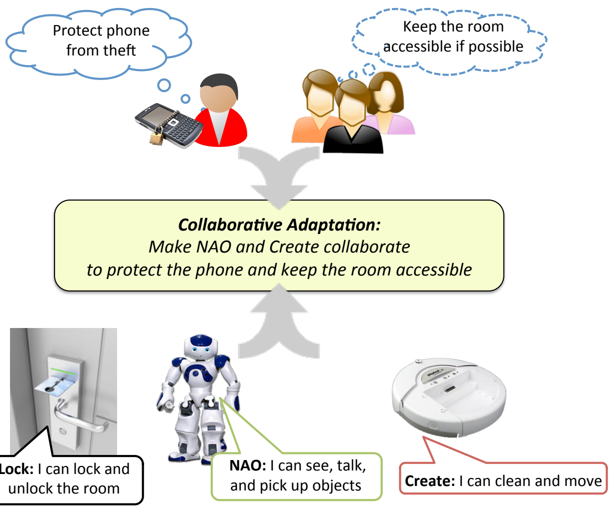

Protect phone from theft
Keep the room accessible if possible
Collaborative Adaptation: Make NAO and Create collaborate to protect the phone and keep the room accessible
Lock: I can lock and unlock the room
NAO: I can see, talk, and pick up objects
Create: I can clean and move

Fig. 2. Illustration of the collaborative robots case study

and iRobot Create to implement the appropriate security control. Therefore, we must answer the following questions:

— Which security control should be implemented?
— Which capabilities should be selected and how should they be configured in order to realise the chosen security control?
— How to compose the selected capabilities?

2.2. Collaborative Security *à la* Michael Jackson

To describe our approach more precisely, we formalise it using Jackson and Zave's framework for requirements engineering [Jackson and Zave 1995], which makes explicit the relationship between requirements, specifications, and environment properties.

The role of our collaborative security framework is to bridge the gap between the security controls and the behaviour of multiple capabilities. The first step is to determine a specification of a security control that satisfies the requirements in the given environment, which can be formalised as follows.

$$\mathcal{SC}, E \models \mathcal{R}$$

where $\mathcal{SC}$ denotes a specification of a security control, E denotes environment properties, and $\mathcal{R}$ denotes a set of requirements. The security control may need to satisfy multiple requirements. Therefore, the requirements are represented as a partially ordered set $\mathcal{R} = \{R_s, R_1, \ldots, R_m\}$ where R_s denotes security requirements. In this work we assume that security requirements have higher priority but that might not always be the case [Berander and Andrews 2005]. The larger the subset of requirements a security control satisfies, the more appropriate it is. The goal is then to use the available capabilities in order to implement the most appropriate security control.

It might be the case that none of the available capabilities can realise this security control on its own:

$$\forall C \in \mathcal{S} : C \not\models \mathcal{SC}$$

where $\mathcal{S}$ denotes the set of capabilities available in the environment.

Furthermore, even multiple capabilities put together in the environment may not be enough to implement the security control:

$$\forall \mathcal{C} \subseteq 2^{\mathcal{S}} : \mathcal{C} \not\models \mathcal{SC}$$

where $2^{\mathcal{S}}$ denotes the power set of $\mathcal{S}$, i.e. any subset of available capabilities.

Yet by *making* multiple capabilities work together, we can implement this security control. To this end, we automatically synthesise an intermediary software entity, called a *mediator* [Wiederhold 1992], that composes multiple capabilities in order to implement the appropriate security control, which can be formalised as follows:

$$\text{Find } \mathcal{C} \subseteq 2^{\mathcal{S}} \text{ and synthesise } M \text{ such that } \mathcal{C}, M \models \mathcal{SC}$$

where $\mathcal{C}$ is the selected set of capabilities and M is the synthesised mediator. We consider the entailment ($\models$) from two perspectives: features and behaviours. From a feature perspective, the features of the selected capabilities must be sufficient to implement the security control. These features must also optimise specific quality attributes of the implementation of this security control. From a behavioural perspective, the composition of the behaviours of the selected capabilities together with the synthesised mediator must refine the behaviour specified by the security control. This refinement implies the inclusion of the traces (possible executions) of the security control specification into those of the system composed of the selected capabilities together with the synthesised mediator [Clarke and Wing 1996]. Note that the mediator, if it exists, is only responsible for coordinating the behaviours of the selected capabilities rather than creating additional functionalities, i.e. a behaviour with a new set of actions (alphabet) [Bennaceur and Issarny 2015].

Hence, collaborative security aims to realise security controls by dynamically composing the capabilities of the IoT devices available at runtime. After giving the necessary formal definitions in Section 3, Section 4 details feature selection, that is finding $\mathcal{C} \subseteq 2^{\mathcal{S}}$. Section 5 details feature-driven mediator synthesis, that is synthesising M.

3. PRELIMINARIES

In this section, we give the formal definitions of the models used within our collaborative security framework.

3.1. Modelling Requirements and Security Controls using KAOS

In this section we show how feature and behavioural models can be used to represent the security controls necessary to satisfy security requirements as well as other relevant requirements. We build upon KAOS goal modelling [van Lamsweerde 2009] to represent and reason about the relationships between requirements and security controls. A KAOS goal model shows how goals are *refined* into sub-goals and associated domain properties. A KAOS *goal* is defined as a prescriptive statement that the system should satisfy through the cooperation of agents such as humans, devices and software. Goals may refer to services to be provided (functional goals) or quality of service (soft goals). KAOS domain properties are descriptive statements about the environment. Besides describing the contribution of sub-goals (and associated domain properties) to the satisfaction of a goal, refinement links are also used for the operationalisation of goals. In this case, refinement links map the goals to *operations*, which are atomic tasks executed by the agents to satisfy those goals. *Conflict* links are used to represent the case of goals that cannot be satisfied together. Keywords such as Achieve, Maintain, and Avoid are used to characterise the intended behaviours of the goals and can guide their formal specification. A KAOS requirement is defined as a goal under the responsibility of a single software agent.

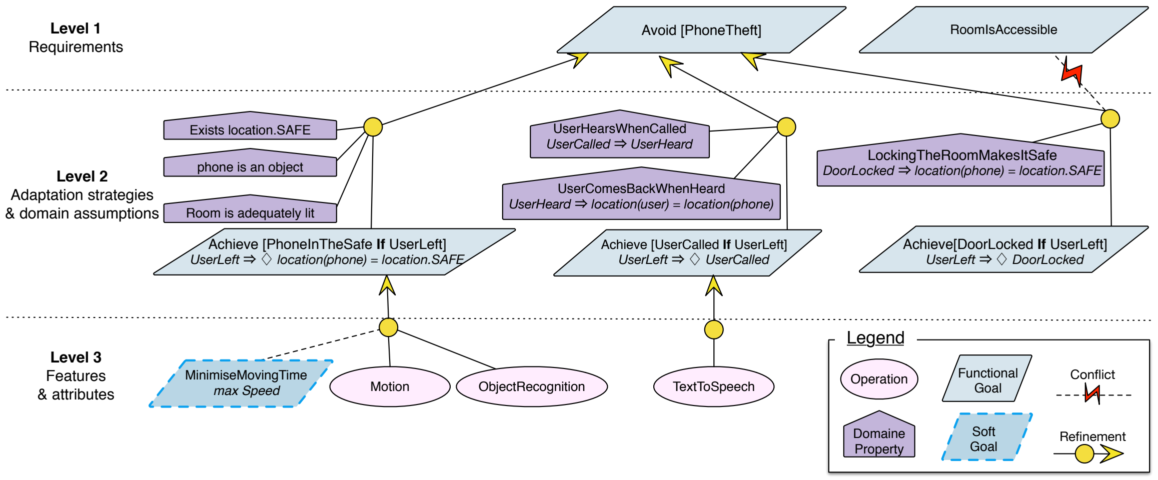


Fig. 3. Refining the requirements of the collaborative robots case study

As our starting point, we consider security (and other) requirements, all of which are under the responsibility of the collaborative security framework, which composes multiple capabilities in order to satisfy these requirements. The goal model used to refine requirements into features and behavioural models can be decomposed into three levels. The first level specifies the requirements. The second level is dedicated to security controls. It defines the behavioural specification and the domain properties associated with each security control. The third level is about the features necessary to implement the security controls as well as the attributes that must be optimised by the implementation. Note that the use of levels facilitates the representation but does not have a formal groundings.

Fig. 3 depicts the goal model for the collaborative robots case study. Two requirements are specified $\mathcal{R} = \{R_s, R_{usability}\}$ where R_s is to protect the phone from theft and $R_{usability}$ is to keep the room accessible. Refinement links capture alternative security controls that satisfy those requirements assuming some domain properties. Conflicts links may interconnect security controls and other requirements and help select the appropriate security control to implement. For example, the security control that involves moving the phone to a safe place contributes to the satisfaction of both the security and usability requirements whereas the security control for locking the room also satisfies the security requirement but not the usability one. Each security control is annotated using a Linear Temporal Logic (LTL) [Pnueli 1977] formula that specifies the desired behaviour of this security control. For example, the security control that involves moving the phone into a safe place specifies that when the user leaves the room then the location of the phone shall eventually become the safe place, i.e. G_s is defined as $UserLeft \Rightarrow \Diamond location(phone) = location.SAFE$. This security control is associated with three domain properties: *phone is an object*, *there exists a safe place*, and *the room is adequately lit*. Finally, each security control is refined into a set of features necessary to implement this security control as well as the attributes that need to be optimised by this implementation. For example, the security control that involves moving the phone into a safe place necessitates two features Motion and ObjectRecognition and shall maximise the Speed attribute. Note that the KAOS model specifies possible security controls but is agnostic about existing available capabilities, which are then selected and configured by the collaborative security framework in order to realise the specified security control.

3.2. Modelling Capabilities using Featured Transition Systems

Capabilities describe what an IoT device can do, i.e. its features, and how it interacts with the environment, i.e. its behaviour. This section gives the formal definitions of the models used to represent capabilities.

A Feature Model (FM) is a hierarchical organisation of features representing the constraints under which features occur in valid configurations [Kang et al. 1990].

Definition 3.1. A **feature model** is a tuple $FM = (\mathcal{F}, DE, G, Car, r, Attr, \rho, \tau, \Phi)$ where $\mathcal{F}$ is a finite set of features, $DE \subseteq \mathcal{F} \times \mathcal{F}$ is a set of directed child-parent edges, $G \subseteq 2^{DE}$ are non-overlapping sets of edges participating in feature groups, i.e. the edges of the same sets share the same parent, $Car : G \rightarrow \mathbb{N}_0 \times \mathbb{N}_0$ is a mapping from a group to a pair $(i..j)$ denoting the cardinality of the group where i is the minimum number of children required and j the maximum, $r \in \mathcal{F}$ is the root feature, $Attr$ is a set of attributes, $\rho : Attr \rightarrow \mathcal{F}$ is a total function that associates an attribute with a feature, $\tau : Attr \rightarrow \{Integer, Real, Boolean, Enumeration\}$ assigns a type to each attribute. This type must be finite or an interval of real numbers, Φ is a set of boolean-valued expressions over the features $\mathcal{F}$ and the attributes $Attr$, expressing constraints on the selection of features.

Unlike the definition of Classen *et al.* [Classen et al. 2011], the root feature is optional. The rationale behind making the root optional is that during feature selection, a capability is selected only if the corresponding root feature is selected. As we will show in the Section 4, the root feature is selected if any of the features associated with this capability is selected.

The behaviour of a device specifies how it interacts with its environment. Transition Systems (TS) are often used to specify behaviours [Keller 1976].

Definition 3.2. A **Transition System (TS)** is a tuple $M = (S, A, Tr, s_0, Pred, L)$ where S is a finite set of states, A is a set of observable actions, $Tr \subseteq S \times A \times S$ denotes a transition relation, $s_0 \subseteq S$ is the initial state, $Pred$ is a set of predicates, and $L : S \rightarrow 2^{Pred}$ is a valuation function that indicates for a state $s \in S$ the predicates $p \in 2^{Pred}$ that are true in this state.

However, the behaviour of an IoT devices is related to its features. Specifically, the invocation of the actions of A is conditioned by the enabled features. Featured Transition Systems (FTS) provide a compact formalism for describing behaviours using feature models [Classen et al. 2013]. An FTS is a TS whose actions are guarded by features, which are specified in a feature model.

Definition 3.3. A **Featured Transition System (FTS)** is a tuple $B = (S, A, Tr, s_0, Pred, L, FM, \gamma)$ where $S, A, Tr, s_0, Pred$, and L are defined as in Definition 3.2, FM is a feature model, and $\gamma : Tr \rightarrow \mathbb{B}(\mathcal{F})$ is a total function, labelling each transition $t \in Tr$ with a feature expression $b \in \mathbb{B}(\mathcal{F})$ that must be true for the action associated with t to be executed. $\mathcal{F}$ are the features associated with the feature model FM.

For a selected set of features (i.e. a configuration), an FTS can be projected onto this configuration by removing all the transitions whose feature expressions are not satisfied, which results in a TS.

Definition 3.4. A **projection** of an FTS B onto a set of features $f \subseteq \mathcal{F}$ is a TS $B_{|f} = \big(S, A, Tr', s_0, Pred, L, FM, \gamma\big)$ where $Tr' = \{t \in Tr | f \models \gamma(t)\}$.

Definition 3.5. A **capability** is an FTS B describing the behaviour and the features $\mathcal{F}$ of an IoT device.

4. FEATURE SELECTION AS A MULTI-OBJECTIVE CONSTRAINED OPTIMISATION PROBLEM

The first step in achieving collaborative security is to find the subset of capabilities that need to be composed in order to implement the appropriate security control. These capabilities are also configured by enabling only the features required for this collaboration and which optimise the implementation of this security control. The selection of this optimal set of features is the focus of this section. Fig. 4 gives an overview of feature selection. We are provided with a set of feature models, each of which associated with a capability, together with the specification of a security control. The specification of the security control includes the set of features necessary for its implementation and the quality attributes that need to be optimised by this implementation. The aim is to select a set of features to be enabled which (i) includes all the features of the security control, (ii) satisfies the constraints imposed by the feature model of each capability, and (iii) optimises the quality attributes for the implementation of the security control. We formulate feature selection as a MOCOP (see Definition 4.1). By doing so, we can build upon the large body of work on solving optimisation problems efficiently using constraint programming [Rossi et al. 2006].

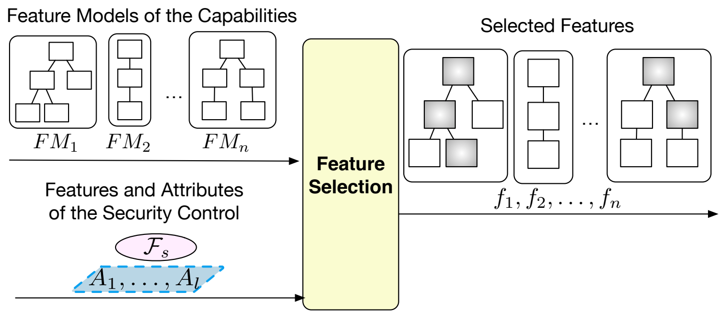


Fig. 4. Feature selection

Constraint programming uses constraints to state the problem declaratively without specifying a computational procedure to solve it. The latter task is carried out by constraint solvers. The constraint solver implements intelligent search algorithms such as backtracking and branch and bound which are exponential in time in the worst case but that have proved to be very efficient in practice. The constraint solver also exploits the arithmetic properties of the operators used to express the constraints to quickly check, discredit partial solutions, and prune the search space substantially.

Definition 4.1. A **Multi-Objective Constrained Optimisation Problem (MOCOP)** is a tuple (X, D, T, U) where $X = \{x_1, ..., x_n\}$ is the set of variables of the problem; D is a function that associates to each variable x_i its domain $D(x_i)$, i.e. the set of possible values that can be assigned to x_i; $T = \{T_1, ..., T_m\}$ is the set of constraints. A constraint T_j is a mathematical relation defined over a subset $x^j = \{x^j_1, \ldots, x^j_{n^j}\} \subseteq X$ of variables, which restricts the values that these variables can take at the same time; and $U = \{U_1, ..., U_k\}$ is a set of objective functions whose values we seek to optimise. An objective function $U_{l=1..k}$ is defined over a subset of variables $Y \subseteq X$ and associates a utility—usually an integer or real value—to each assignment of Y.

To formulate feature selection as a MOCOP, we must define the variables X and their domains $D(X)$, the associated set of constraints T, and the objective functions U.

Variables and their Domains. Feature selection involves searching among all possible combinations of features, an optimal set of features that must be enabled on a subset of capabilities in order to implement a security control. Therefore, we associate each capability with a variable describing whether the features of this capability are enabled or not. More specifically, we represent each variable as a vector of boolean values, each of which set to 1 (true) if the feature of the associated capability is to be enabled and set to 0 (false) otherwise:

$$X = \{x_1, x_2, \ldots, x_n\} \text{ and } D(X) = 2^{\mathcal{F}_1} \times 2^{\mathcal{F}_2} \times \ldots \times 2^{\mathcal{F}_n}$$

where $2^{\mathcal{F}_{i=1..n}}$ denotes the power set of the features of capability $C_{i=1..n}$.

Constraints. Constraints specify the conditions for selecting features. In the following, the first constraint ensures that all the features of the security control are selected and the second concerns conformance to the feature models of individual capabilities.

Constraint 1: The selected features must include the features necessary to implement the security control. This constraint ensures that each feature required to realise the security control is selected in some capability and is formalised as follows:

$$\forall f \in \mathcal{F}_s, \exists\, k \in [1, n] \text{ such that } x_k[f]$$

where $\mathcal{F}_s$ denotes the set of features associated with the security control and $x_{k=1..n}$ is the variable associated with capability $C_{k=1..n}$.

Constraint 2: The selected features must conform to the feature model of the capabilities. The features to enable on each capability must represent a valid configuration considering the feature model of this capability. We describe below how this constraint is decomposed into smaller constraints that must be respected for each capability.

— The selected features must satisfy the cardinality of every group of features. Whenever a parent feature with a cardinality $<a..b>$ is selected, then at least a children must be selected, i.e. the number of child features that are true (set to 1) is at least a, and at most b, which can be formalised as follows.

$$\forall f_j \in \mathcal{F}_i \text{ such that } gr \in G_i, gr = \{(f_{j_1}, f_j), \ldots, (f_{j_m}, f_j)\} \text{ and } < a..b >= Car_i(gr),$$
$$x_i[f_j] \Rightarrow \left(a \leqslant \sum_{k=1}^{m} x_i[f_{j_k}] \leqslant b\right) \quad \text{for } 1 \leqslant i \leqslant n$$

where $f_j \in \mathcal{F}_{i=1..n}$ denotes the parent feature of $f_{j_1}, \ldots, f_{j_m}$, i.e. $gr = \{(f_{j_1}, f_j), \ldots, (f_{j_m}, f_j)\}$ is a group $gr \in G_{i=1..n}$ associated with the capability $C_{i=1..n}$. The cardinality of this group is $<a..b>$.

— Whenever a child feature is selected, so is the parent feature. As a result, the root feature, and the capability, is selected if any of its features is selected.

$$\forall (f_{child}, f_{parent}) \in DE_i \ : \ x_i[f_{child}] \Rightarrow x_i[f_{parent}] \quad \text{for } 1 \leqslant i \leqslant n$$

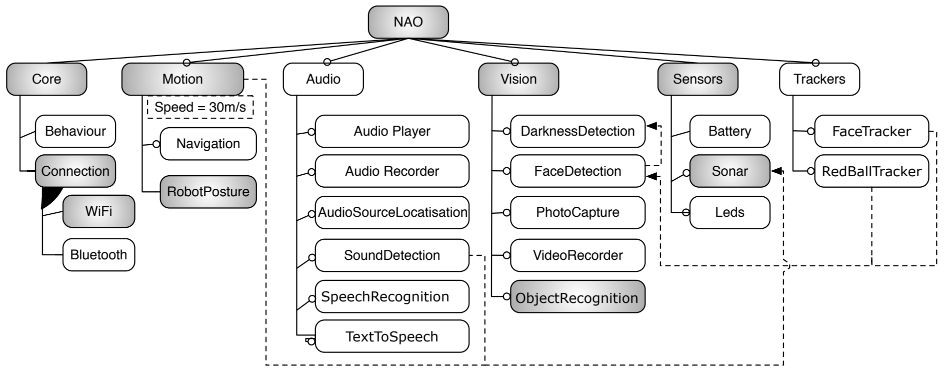
where $(f_{child}, f_{parent}) \in DE_{i=1..n}$ is a child-parent pair associated with the feature model of capability $C_{i=1..n}$.

— The selection of features must conform to any additional condition on features and attributes specified within the feature model of the capability:

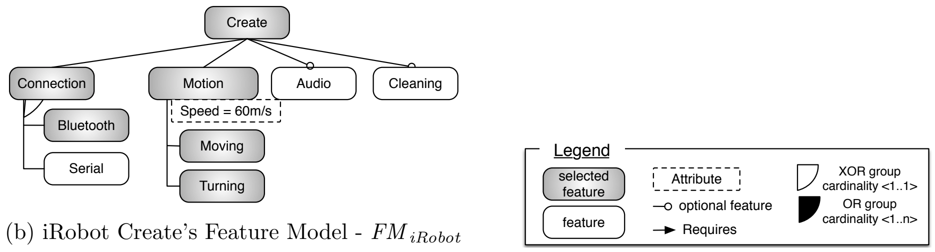
$$\Phi_i \quad \text{for } 1 \leqslant i \leqslant n$$

where $\Phi_{i=1..n}$ is the boolean expression associated with the feature model of capability $C_{i=1..n}$.

Objective functions. There might be multiple sets of features satisfying the aforementioned constraints. Therefore, the search is driven by the quality attributes we seek to optimise. In addition, the number of selected features must also be minimised.



(a) NAO's Feature Model - FM_{NAO}



(b) iRobot Create's Feature Model - FM_{iRobot}

Fig. 5. Feature models for the collaborative robots case study

We assume without loss of generality that we seek to minimise all quality attributes without a preference for any of these attributes, which can be stated as follows.

$$\min_{s \in D(X)} [g_{A_1}(s), g_{A_2}(s), \ldots, g_{A_l}(s), |s|]$$

where $g_{A_{j=1..l}}$ denotes the function that calculates the value of the quality attribute $A_{j=1..l}$ for any set of features $s \in D(X)$ that satisfies the aforementioned constraints. $|s|$ denotes the cardinality of this set of features. For simplicity, we assume that the attribute is associated with a single selected feature. When an attribute $A_{j=1..l}$ spans multiple selected features, then $g_{A_{j=1..l}}$ is an aggregation function (e.g., summation, multiplication, and minimum) that depends on the structure of the feature model and the type of attribute, in a way similar to QoS-aware service composition [Zeng et al. 2004; Jaeger et al. 2005; Tan et al. 2016].

Example

Let us consider the collaborative robots case study, Fig. 5 depicts the feature models of NAO and iRobot Create. Note that Definition 3.1 describes the feature model in a normal form. Therefore, AND, OR, and XOR groups are represented using the cardinalities, $<n..n>$, $<1..n>$, and $<1..1>$ respectively. The *Requires* links are represented as logical implications within the set of boolean-valued expressions Φ. The greyed features represent the optimal set of features that need to be selected in order to implement the security control that involves moving the phone to a safe place. This security control necessitates the Motion and ObjectRecognition features (see Fig. 3), both of which are selected. The selection of features must also conform to the cardinality of each group. For example, in the feature model of iRobot Create, Connection admits only one child feature, which in this case is the Bluetooth feature. In addition, as we want to maximise the speed at which the phone is moved to a safe place, the Motion feature of iRobot Create whose associated Speed has a value of 60 m/s is selected instead of the Motion feature of NAO whose associated Speed has only a value of 30 m/s.

Complexity

We prove that feature selection is NP-complete using polynomial-time reductions from the set cover problem [Karp 1972]. We recall that in the set cover problem, we are given a set of n elements U and a finite family of its subsets $C = \{S_1, \ldots, S_m\}$ such that $S_i \subseteq U$ and $\bigcup_{i=1}^{m} S_i = U$, we must find a minimum-cost collection of these subsets whose union is U, i.e. the smallest (k) family of subsets $C' = \{T_1, \ldots, T_k\}$ such that $T_j \in C$ and $\bigcup_{j=1}^{k} T_j = U$.

The first step is to transform an instance of the set cover problem into an instance of capability selection. We start by building m feature models, each of which associated with a subset S_i. The feature model consists of a root feature r_{S_i} and the conjunction of its mandatory children features, which are the elements included in S_i. Hence, $\mathcal{F}_i = S_i \cup \{r_{S_i}\}$. In addition, each root feature r_{S_i} is associated with an attribute $cost = 1$. The features associated with the security controls are the n elements of U, i.e. $\mathcal{F}_s = U$. We seek then to minimise the $cost$ attribute, i.e. $A = cost$. The function g_{cost} for computing the value of the attributes for a set of feature is the sum of the values of the $cost$ attribute for individual features. It is trivial that this translation of the set cover problem to an instance of feature selection can be performed in polynomial time.

Feature selection computes the optimal set of features $f_1, f_2, \ldots, f_m$ that includes all the features of the security control and minimises $cost$. In addition, if a child feature is selected, so is the parent feature r_{S_i} since the selected set of features must satisfy the individual feature models. Hence to get a solution to the set cover, it suffices to pick the set S_i whose associated root feature, r_{S_i} , is enabled. It is also trivial that we can check the satisfaction of feature models and inclusion of the features of the security controls in polynomial time. As a result, we can state that feature selection is NP-complete.

5. FEATURE-BASED MEDIATOR SYNTHESIS

Features express only the functionality associated with a capability. To ensure that capabilities can be composed in order to implement the appropriate security control, we need to reason about their behaviours and compose them appropriately. This composition is ensured through mediator synthesis. Fig. 6 gives an overview of feature-driven mediator synthesis. We are provided with the set of selected features and the behavioural models of the of the associated capabilities together with the behavioural specification of a security control. The goal is to synthesise, if possible, a mediator that coordinates the behavioural models of the capabilities in order to satisfy the behavioural specification of the security control.

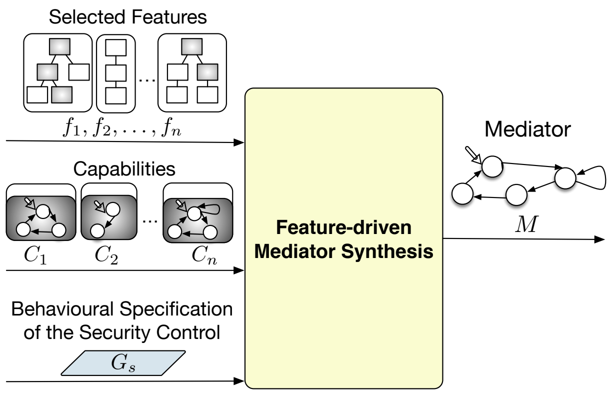


Fig. 6. Feature-driven mediator synthesis

There are many approaches to mediator synthesis [Yellin and Strom 1997; Vaculín et al. 2009; Mateescu et al. 2012; Cavallaro et al. 2012; Inverardi and Tivoli 2013; D'Ippolito et al. 2013; Bennaceur and Issarny 2015]. These approaches differ in their assumptions (e.g., deterministic behaviour or a provided partial specification of the mediator) and the expressiveness of the goals involved (e.g., dealing with safety, liveness, or general LTL properties). All these approaches require the exploration of the state space of the composition of behaviours, which can rapidly grow as the number of capabilities increase.

To reduce the state space to be explored by the synthesis algorithm, we propose to *project* the behaviours represented by each capability onto the selected features. Projection scopes the behaviours of individual capabilities by keeping only the transitions whose feature expressions are satisfied. We recall that the projection of an FTS onto a set of features is the TS obtained by removing all the transitions whose feature expressions are not satisfied (see Definition 3.4). In addition, this TS is well-formed in that it does not contain any dead state, i.e. a state with no outgoing transitions. Hence, we can project the behaviours represented by the capabilities onto the selected set of features to obtain their actual behaviours when only these features are enabled. We then seek to synthesise, if possible, a mediator such that the composition of the projected behaviours together with the mediator satisfies the behaviour specified by the security control, which can be formally specified as follows.

$$C_{1|f_1} \parallel C_{2|f_2} \parallel \ldots C_{n|f_n} \parallel M \models G_s$$

where $f_1, f_2, \ldots, f_n$ denote the selected set of features, G_s an LTL property specifying the desired behaviour of the security control, and M the synthesised mediator. The algorithm for mediator synthesis (Algorithm 1) starts by checking the basic configuration where G_s is satisfied in the initial states (Lines 1-5). The algorithm then systematically explores the state space of the composition of the projected capabilities (C_i for simplicity) to synthesise a mediator (Lines 6-20). This exploration is guided by the selection of a capability whose initial state s_{0_k} will be further examined (Line 7). The selection of a capability to explore may be random, sequential (the same index until all states have been explored), or motivated by some heuristics related to the likelihood of satisfying G_s but, for instance, let us assume that the algorithm simply loops from 1 to n, which results in a breadth-first exploration. To avoid cycles and loops, s_{0_k} is deleted from the state set (Line 8). For each outgoing transition (s_{0_k}, a_k, s_k), an updated TS is created by removing the transition and setting s_k as the initial state (Line 10). A recursive call with the updated TS together with the remaining capabilities is then made to test whether this transition can lead to a valid mediator (Line 11). Note that we present the recursive version of the algorithm for readability while the implementation is based on stacks. In the case where a state satisfying G_s is reachable, the mediator is generated by prefixing the partial mediator with the selected transition (Lines 12-17). The algorithm fails if all states have been explored without reaching a state where G_s is satisfied (Line 21).

If the synthesis algorithm fails to produce a mediator that coordinates the projected behaviours in order to satisfy the goal G_s, then another solution $s' = \{f'_1, f'_2, \ldots, f'_n\}$ is selected and the synthesis algorithm run again. Currently, the selection of features is only informed that no mediator can be synthesised. In future work, we will investigate how the synthesis algorithm can guide the selection of another set of features. If all valid sets of features, i.e. sets of features satisfying the constraints described in Section 4, have been explored then our collaborative security approach cannot realise the chosen security control given the available capabilities and another security control must be chosen. If a mediator can be synthesised for the selected set of features then the security control is successfully implemented thereby satisfying the requirements.

ALGORITHM 1: SynthesiseMediator

```
Input: Projected capabilities: $C_{i=1..n} = (S_i, A_i, Tr_i, s_{0_i}, Pred_i, L_i), G_s$
Output: Mediator: $M$
if $\bigwedge_{i=1}^{n} L_i(s_{0_i}) \not\models false$ and $\bigwedge_{i=1}^{n} L_1(s_{0_i}) \models G_s$ then
    $s \leftarrow NewState()$;
    $M \leftarrow \left(\{s\}, \emptyset, \emptyset, \{s\}, \bigcup_{i=1}^{n} Pred_i, L(s) = \bigwedge_{i=1}^{n} L_i(s_{0_i})\right)$;
    return $M$;
end
while $\bigcup_{i=1}^{n} S_i \neq \emptyset$ do
    $k \leftarrow SelectCapability()$;
    $S_k \leftarrow S_k \backslash \{s_{0_k}\}$;
    for $t : (s_{0_k}, a_k, s_k) \in Tr_k$ do
        $C' \leftarrow (S_k, A_k, Tr_k \backslash \{t\}, s_k, Pred_k, L_k)$;
        $M' = (S', A', Tr', s'_0, Pred', L') \leftarrow SynthesiseMediator(C_{i=1..n,\ i \neq k}, C', G_s)$;
        if $M' \neq$ Nil then
            $s'' \leftarrow NewState()$;
            $M \leftarrow (S' \cup \{s''\}, A' \cup \{a_k\}, Tr' \cup \{(s'', a_k, s'_0)\}, \{s''\}, Pred' \cup L_k(s_{0_k}), L'')$
                where $L'' : S' \cup \{s''\} \rightarrow 2^{Pred' \cup L_k(s_{0_k})}$ such that $L''(s'') = L_k(s_{0_k})$ and
                $L''(s) = L'(s) \wedge L_k(s_{0_k})$ for $s \in S'$;
            return $M$;
        end
    end
end
return Nil;
```

Example

Let us consider the collaborative robots case study. Fig. 7 and 8 depict the behaviours of NAO and iRobot Create respectively. Once the features are selected, the behaviour is projected and some transitions are removed. For example, the transitions say(text) in NAO's behaviour is removed as the guarding features, TextToSpeech and will not enabled. The security control that involves moving the phone to a safe place is specified by the LTL formula $\Diamond location(phone) = location.SAFE$. A mediator must then be synthesised which coordinates the projected behaviours of NAO and iRobot Create (see Fig. 5) in order to reach a state where the expression $location(phone) = location.SAFE$ is true. This mediator is depicted in Fig. 9. The actions are prefixed with the capability names to avoid ambiguity. The mediator starts by invoking the connect actions on both capabilities. Note that it is also possible to start by connecting to iRobot Create but in any case the synthesis algorithm chooses only one sequence of invocations to ensure that the mediator is deterministic. The mediator can then invoke the actions locate(phone) and pick(phone) based on the assumption that phone is an object. After

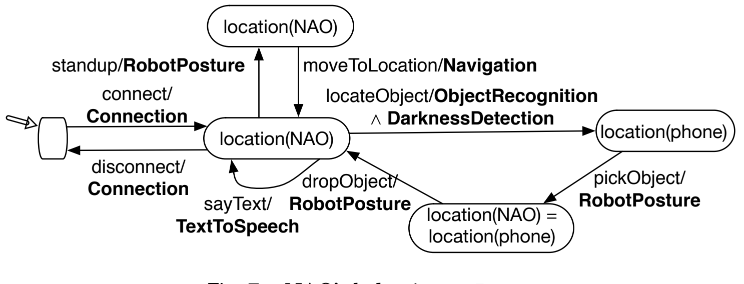

location(NAO)
standup/**RobotPosture**
moveToLocation/**Navigation**
connect/ **Connection**
locateObject/**ObjectRecognition** ∧ **DarknessDetection**
location(NAO)
location(phone)
disconnect/ **Connection**
dropObject/ **RobotPosture**
sayText/ **TextToSpeech**
pickObject/ **RobotPosture**
location(NAO) = location(phone)

Fig. 7. NAO's behaviour - B_{NAO}

the execution of the move(location((NAO))) action by iRobot Create, its location becomes the location of NAO, and the location of the phone once the drop(phone) action is executed. Finally, the phone is moved to a safe place once iRobot Create executes the action move(location.SAFE)). The subsequent actions are executed to return to the initial states.

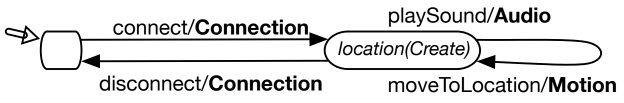


Fig. 8. iRobot Create's behaviour - B_{iRobot}

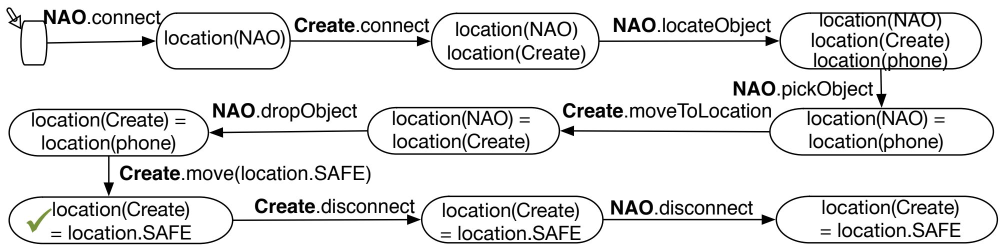

NAO.connect
location(NAO)
Create.connect
location(NAO) location(Create)
NAO.locateObject
location(NAO) location(Create) location(phone)
NAO.pickObject
location(NAO) = location(phone)
Create.moveToLocation
location(NAO) = location(Create)
NAO.dropObject
location(Create) = location(phone)
Create.move(location.SAFE)
location(Create) = location.SAFE
Create.disconnect
location(Create) = location.SAFE
NAO.disconnect
location(Create) = location.SAFE

Fig. 9. NAO-iRobot Create Mediator

Complexity

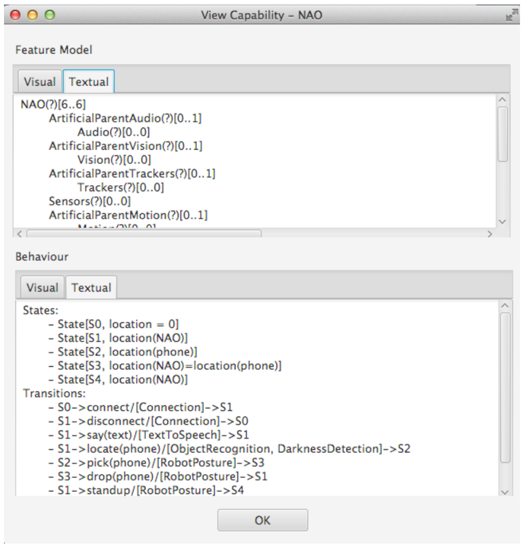
In the general case, mediator synthesis is known to be computationally expensive [Pnueli and Rosner 1989]. Let us consider the synthesis of a mediator that ensures the goal G_s assuming E, i.e. the mediator satisfies the formula $\phi = E \Rightarrow G_s$. When ϕ is expressed as an LTL formula, mediator synthesis may reach complexity of double exponent in the size of ϕ [Pnueli and Rosner 1989]. Yet for safety formulas as well as subclasses of liveness formulas (e.g., GR(1) [Ehlers 2011] or SGR(1) [D'Ippolito et al. 2013]), the synthesis problem can be solved in polynomial time. Our approach does not aim to improve the synthesis algorithm *per se*. Instead, we rely on the extensive work that has been developed in the area of mediator synthesis and reduce the size of the models provided as input to the synthesis algorithm. More specifically, by projecting the behaviour associated with the capabilities, we reduce the size of E and hence the size of ϕ. Furthermore, by simplifying ϕ through projection, we may make it possible to use polynomial-time techniques to synthesise the mediator.

Finally, most techniques for mediator synthesis do not consider the optimisation of numerical attributes of the synthesised mediator. Indeed, controller synthesis for more complicated models (e.g., probabilistic systems) is NP-hard [Baier et al. 2004].

6. VALIDATION

In this section we present a prototype tool, FICS (Feature-driven medIation for Collaborative Security), that implements our approach. We report the results of experiments using FICS to generate mediators for the collaborative robots case study. Finally, we discuss the limitations and possible enhancements of our framework. Our evaluation covers the following properties of our approach:

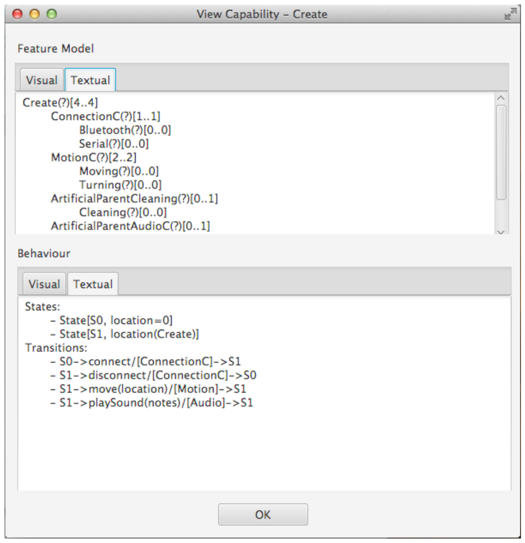
— *Feasibility.* We provide tool support for composing feature-based capabilities and use it in the context of the collaborative robots case study to configure and mediate the capabilities of NAO and iRobot Create automatically.



```
NAO(?)[6..6]
      ArtificialParentAudio(?)[0..1]
            Audio(?)[0..0]
      ArtificialParentVision(?)[0..1]
            Vision(?)[0..0]
      ArtificialParentTrackers(?)[0..1]
            Trackers(?)[0..0]
      Sensors(?)[0..0]
      ArtificialParentMotion(?)[0..1]
```

```
States:
      - State[S0, location = 0]
      - State[S1, location(NAO)]
      - State[S2, location(phone)]
      - State[S3, location(NAO)=location(phone)]
      - State[S4, location(NAO)]
Transitions:
      - S0->connect/[Connection]->S1
      - S1->disconnect/[Connection]->S0
      - S1->say(text)/[TextToSpeech]->S1
      - S1->locate(phone)/[ObjectRecognition, DarknessDetection]->S2
      - S2->pick(phone)/[RobotPosture]->S3
      - S3->drop(phone)/[RobotPosture]->S1
      - S1->standup/[RobotPosture]->S4
```

Fig. 10. NAO capability in FICS



```
Create(?)[4..4]
      ConnectionC(?)[1..1]
            Bluetooth(?)[0..0]
            Serial(?)[0..0]
      MotionC(?)[2..2]
            Moving(?)[0..0]
            Turning(?)[0..0]
      ArtificialParentCleaning(?)[0..1]
            Cleaning(?)[0..0]
      ArtificialParentAudioC(?)[0..1]
```

```
States:
      - State[S0, location=0]
      - State[S1, location(Create)]
Transitions:
      - S0->connect/[ConnectionC]->S1
      - S1->disconnect/[ConnectionC]->S0
      - S1->move(location)/[Motion]->S1
      - S1->playSound(notes)/[Audio]->S1
```

Fig. 11. iRobot Create capability in FICS

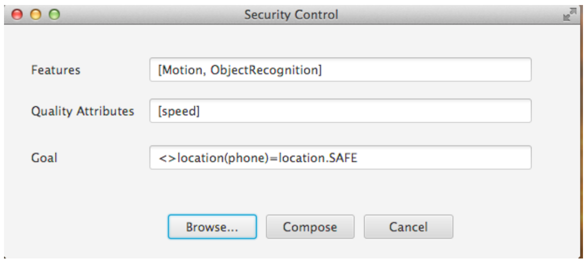
— *Performance.* We measure the time to perform the selection of features and synthesise mediators in the collaborative robots case study to show that, although theoretically complex, feature-based composition can be applied at runtime in practical cases.
— *Scalability.* We show that the use of feature selection can allow mediator synthesis to deal with an increasing number of capabilities.

We focus on evaluating the composition of capabilities to implement security control and its applicability at runtime rather than how to monitor domain properties or when to trigger the composition. We refer the interested reader to related work [Manna et al. 2013] for an analysis of when to trigger the composition.

6.1. Implementation

In order to demonstrate the validity of our approach, we implemented the FICS tool and made it available at `http://sead1.open.ac.uk/fics/`. FICS takes as input the specifications of a set of capabilities, each of which provided by the developer of the IoT device, and that of a security control, provided by a security expert. Note that the tool is equipped with a graphical interface for illustration since the aim is to use some discovery protocol (e.g., UPnP [Jeronimo and Weast 2003]) to detect the available capabilities and compose them automatically (see Fig. 1). A capability is added to the capability set by loading its feature model, described using TVL [Classen et al. 2011], and the associated behaviour, described in a proprietary XML format. We built upon the TVL parser provided by Classen and colleagues and available at `http://projects.info.unamur.be/tvl/` to extract the normal form of the feature model. We updated the TVL parser to support the analysis of numerical attributes for feature models in order to manage the quality attributes to be optimised. During the parsing of the behavioural specification, we check that all actions are guarded by feature expressions described in the associated feature model. Fig. 10 and 11 illustrate the NAO and iRobot Create capabilities once they have been parsed while their TVL specifications are available on the FICS website.

The security control is described using features and attributes, which conform to the names of the features and attributes used to represent the capabilities as well as an



Security Control

Features: [Motion, ObjectRecognition]

Quality Attributes: [speed]

Goal: <>location(phone)=location.SAFE

Fig. 12. Specifying a security control in FICS

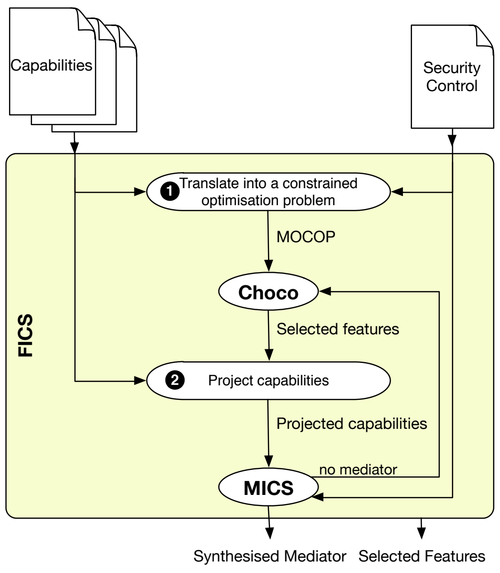
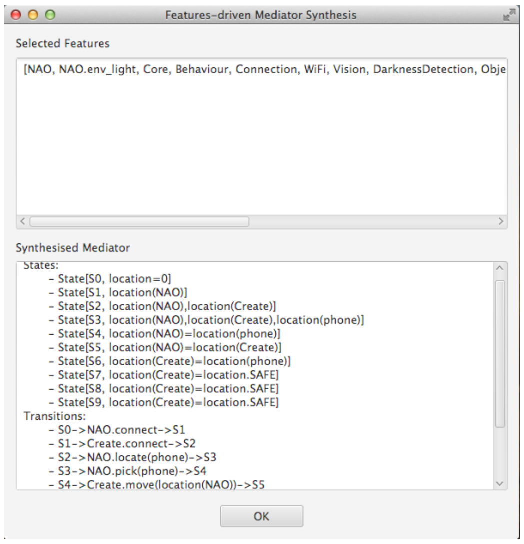


Fig. 13. Overview of FICS



Features-driven Mediator Synthesis

Selected Features

[NAO, NAO.env_light, Core, Behaviour, Connection, WiFi, Vision, DarknessDetection, Obje

Synthesised Mediator

```
States:
      - State[S0, location=0]
      - State[S1, location(NAO)]
      - State[S2, location(NAO),location(Create)]
      - State[S3, location(NAO),location(Create),location(phone)]
      - State[S4, location(NAO)=location(phone)]
      - State[S5, location(NAO)=location(Create)]
      - State[S6, location(Create)=location(phone)]
      - State[S7, location(Create)=location.SAFE]
      - State[S8, location(Create)=location.SAFE]
      - State[S9, location(Create)=location.SAFE]
Transitions:
      - S0->NAO.connect->S1
      - S1->Create.connect->S2
      - S2->NAO.locate(phone)->S3
      - S3->NAO.pick(phone)->S4
      - S4->Create.move(location(NAO))->S5
```

Fig. 14. Visualising the selected features and corresponding synthesised mediator in FICS

LTL formula representing its desired behaviour. Fig. 12 illustrates how the security control that involves moving the phone to a safe place is specified in FICS.

The capabilities available are then automatically composed in order to implement the specified security control. Fig. 13 gives an overview of the FICS tool. The first step is to formulate feature selection as a MOCOP as described in Section 4 (see Fig. 13-❶). We use an open-source java library for constraint solving and constraint programming, Choco[3], to solve this MOCOP problem and find the optimal set of features to select.

The next step is to project the capabilities onto the selected features (see Fig. 13-❷). We use the MICS tool[4] to synthesise a mediator that coordinates the projected capabilities in order to refine the behaviour of the security control. Note that other mediator synthesis tools can be used but we chose MICS because we are familiar with the language and data structures and no conversion or translation of the inputs/outputs is needed during the integration. MICS either generates a mediator that coordinates the behaviours given as input in order to realise the behavioural specification of the security control or returns that no such mediator exists. In the former case, the mediator

[3]`http://choco.emn.fr/`

[4]`http://www-roc.inria.fr/arles/software/mics/`

and the selected features to be enabled are given as output. In the latter case, we select the next valid set of features. If no more valid sets of features can be found then FICS is unable to realise the security control given the capabilities provided as input. Fig. 14 illustrates the output of FICS for the collaborative robots case study when specifying the security control that involves moving the phone to a safe place while the capabilities of NAO and iRobot Create are available.

6.2. Experimental Results

This section presents some practical aspects of our approach by reporting on the results of using FICS to synthesise mediators in the collaborative robots case study.

Table I. Processing time (in milliseconds) for each step

	NAO alone ($n = 1$)	**NAO and iRobot Create** ($n = 2$)
$\prod_{i=1}^{n} \lvert\mathcal{F}_i\rvert$	28	28×9
Time for feature selection (ms)	5 ms	6 ms
$\prod_{i=1}^{n} \lvert States(C_i)\rvert$	6	6×3
Time for synthesis (ms)	2 ms	5 ms

Table I reports the time to perform each composition step (feature selection and mediator synthesis) in the cases where only NAO's capability is available and where both the capabilities of NAO and iRobot Create are available. In both cases, the security control involves moving the phone to a safe place and its implementation should optimise the speed at which the phone is moved. In this experiment we consider a setting where the synthesised mediator is deployed on a MacBook Pro laptop with 2.8 GHz Intel Core i7 processor and 8 GB memory. We also configured the heap memory of the JVM to the maximum. In both cases the time for selecting the features and synthesising the mediator is quasi-instantaneous. Yet in the case where both the capabilities of NAO and iRobot Create are available, the speed at which the phone is moved into the safe is higher, thereby optimising the implementation of the security control.

To evaluate the benefit of using features within mediator synthesis empirically, we proceeded with further experiments by increasing the number of capabilities To generate the capabilities, we introduced multiple NAO's capabilities with variable values for the Speed attribute. We also added a Distance attribute representing the distance between NAO and the phone, which we also varied across the capabilities. We then kept the same security control involving moving the phone to a safe place. We measured the time necessary to synthesise a mediator in both the cases where feature selection is used and where it is not. We repeated the synthesis 30 times for each case and computed the mean time.

In the first experiment, the security control does not specify any attribute to optimise. Fig. 15 shows the time for synthesising mediators in relation to the number of capabilities where feature selection is used and where it is not. The mediation with feature selection does not go beyond 5 capabilities as the state space reaches $28^5 = 17210367$ possible configurations. Without any optimisation, the number of valid sets of features is very high and the feature selection tries to compute all solutions before invoking the mediator synthesis. One way around this is to use the constraint solver to find one valid set of features and then invoke the synthesis. If the synthesis

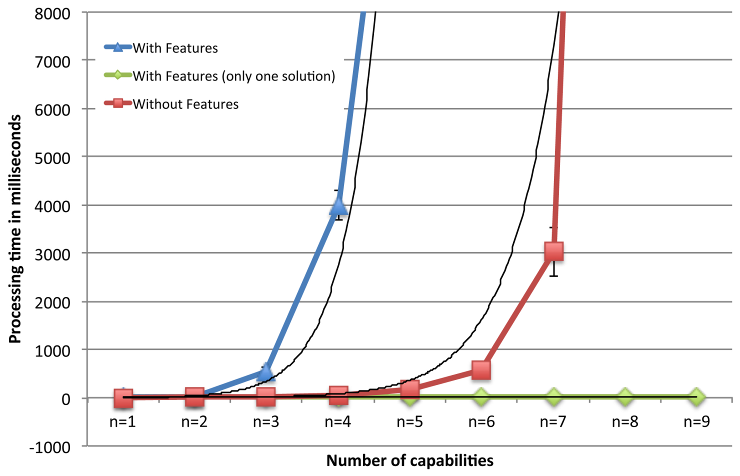


Fig. 15. Processing time according to the number of capability (without attribute optimisation)

does not succeed then the constraint solver resumes and provides another solution. This process allows the synthesis to perform better as shown in Fig. 15. Feature-based mediation is penalised in this case as the number of optional features is high. However, this is the case only when no optimisation is involved.

In the second experiment, the security control specifies one attribute to optimise (mono-objective) or two attributes (multi-objective). Fig. 16 shows the time for synthesising mediators according to the number of capabilities when using feature selection with two attributes (both Speed and Distance) or a single attribute (only Speed) as well as without feature selection. While the processing time always increases proportionally to the number of capabilities, it is much slower when considering feature selection. The time for mediator synthesis without using feature selection or using feature selection with multi-objective optimisation can be fitted to an exponential curve but the former soars with much fewer capabilities. In the case of feature selection with single-attribute optimisation, the curve can be fitted to a third-order polynomial curve. As the search for the set of feature is directed by the attribute to optimise, the algorithm quickly converges to the single set of features optimising the Speed attribute. During mediator synthesis, only the behaviour associated with a single capability has to be explored. Without feature selection, mediator synthesis runs out of memory with 10 capabilities. Indeed, the state space is $6^{10} = 60466176$. Although on-the-fly reduction (e.g., [Mateescu et al. 2012]) can be used, the mediator synthesis algorithm still needs to explore a large state space, which rapidly increases as capabilities are added. This evaluation provides preliminary evidence that using feature makes the composition more scalable in common cases. Indeed, it is more likely that there needs to be some optimisation when implementing a security control [Yuan et al. 2014].

6.3. Discussion

Collaborative security aims to realise security controls according to the capabilities available at runtime. Therefore, it can be used to react rapidly to changes in the environment, changes in assets under protection and their values, and the discovery of new threats and vulnerabilities using the capabilities already available. It can also be used to achieve defence in depth [Stytz 2004] by exploiting alternative implementations of security controls. The framework described in this paper presents the core of our collaborative security approach. We made several simplifying assumptions in or-

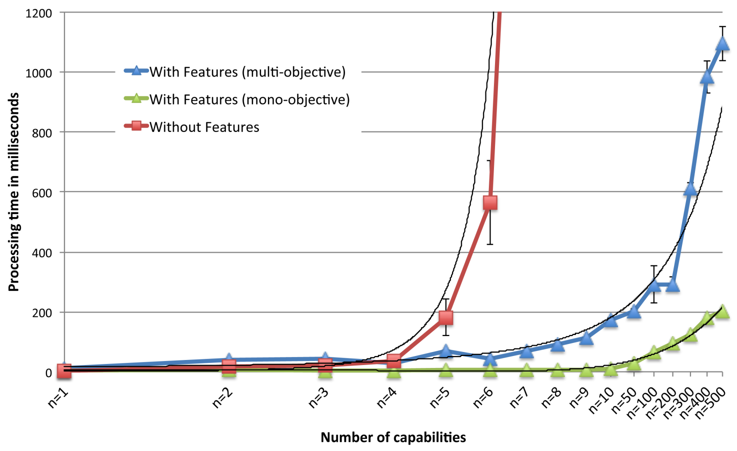


Fig. 16. Processing time according to the number of capability (with attribute optimisation)

der to implement and empirically evaluate this framework. In the following we discuss how some of these assumptions can be relaxed.

Can collaboration be applied to other types of requirements besides security? Our approach for composing feature-based capabilities can be used in a broader and more general context. It can provide a means to opportunistically compose the multiple capabilities in order to satisfy given requirements, be they security-related or not. Yet security exacerbates and opens many issues that make collaboration more challenging. First, security involves a great degree of change, which is not only technical (e.g., the discovery of new attacks and vulnerabilities), but also organisational or business related (e.g., new security policies and business strategies). Reacting to these changes rapidly is paramount and is key to minimising the damage of discovered attacks. By opportunistically using available resources, collaboration offers the possibility to react to change in a timely manner. Assurance is also more challenging when it comes to security. Not only do we need to ensure that security and quality requirements are met but also that no vulnerabilities are introduced. However, it is not easy to prove that the collaborations realised do not create any new vulnerabilities. Our approach can be enhanced by adding constraints to avoid combinations of features that may lead to some vulnerability based on a repository of common vulnerabilities such as Common Vulnerabilities and Exposures (CVE)[5] or Common Vulnerability Scoring System (CVSS)[6]. Likewise, anti-goals (attacker' goals) whose satisfaction must be stopped by the mediator can be integrated. Hence, the formulation of feature selection as a multi-objective constrained optimisation problem makes it easy to extend and improve the solution by simply adding additional constraints.

What are the limitations of the collaborative security framework? In our collaborative security approach we are given the specification of a security control and then seek a subset of capabilities and an associated mediator to implement this security control. Therefore, we do not create or invent security controls based on the capabilities available but only use the capabilities available to implement specified security controls. A possible enhancement is to discover new security controls that were not specified at design time by exploiting security models such as attack trees.

[5]`http://cve.mitre.org`
[6]`http://www.first.org/cvss`

We also assume a shared vocabulary of features when specifying capabilities and security controls. This assumption can be relaxed by attaching semantic annotations to the features and using ontologies to reason about and relate the different features. In this case, the ontology becomes the new shared vocabulary, as demonstrated in our previous work [Bennaceur and Issarny 2015].

In our collaborative security approach, iterations between feature selection and mediator synthesis are independent. We need to investigate more efficient ways to trace back the causes of failure in the synthesis and inform the feature selection. Moreover, we can envision learning from both success and failures of mediator synthesis to guide feature selection. One way is to explore how the different sets of features relate to each other and incrementally synthesise the mediator, in a way similar to the work of Greenyer and colleagues [Greenyer et al. 2013].

Finally, we also assume that individual capabilities are trustworthy and that they implement the capabilities advertised. However, the capabilities may deviate from their specified behaviours either due to faults or malicious intents. We are investigating argumentation as a means to ensure assess the risk and seek alternative collaborations to ensure the satisfaction of some requirements even in the case of misbehaviour [Yu et al. 2015].

What can be the role of humans in collaborative security? Human agents can play an active role in satisfying security requirements, acting as sensors, actuators, or decision makers [Haley et al. 2008]. Yet, human behaviour is more difficult to analyse than the behaviours of software components. The work of Cámara *et al.* [Cámara et al. 2015] provide a formal model to represent and reason about human behaviour in order to develop adaptation strategies involving both human agents (acting as actuators only) and software components. Some parameters for this model (e.g., stress levels) can be obtained using wearable sensors. Nevertheless, a richer model may be necessary to involve human agents as decision makers when implementing security controls.

7. RELATED WORK

With the great potential and opportunities of the IoT come a whole set of challenges of which a complete survey is beyond the scope of this paper. We refer the interested reader to one of the many surveys on the subject [Al-Fuqaha et al. 2015; Atzori et al. 2010]. In this paper we focus in particular on composition and security.

One of the fundamental challenges of the IoT is to compose the capabilities of the plethora of devices available [IERC 2015]. This challenge is exacerbated when heterogeneity spans the application, middleware, and network layers. At the application layer, devices may exhibit disparate data types and operations, and may have distinct business logics. At the middleware layer, they may rely on different communication protocols, which define disparate data representation formats and induce different architectural constraints. At the network layer, data may be encapsulated differently according to the network technology in place. While standardisation efforts such as HyperCat [HyperCat Consortium 2016], AllJoyn [Linux Foundation 2016a], and IoTivity [Linux Foundation 2016b] are suggested as potential solutions at the network and middleware layers, the diversity of IoT applications requires additional effort to deal with semantic interoperability at the application layer [IERC 2015]. In previous work [Bennaceur and Issarny 2015], we developed an approach based on ontology reasoning and constraint programming to synthesise application-layer mediators automatically. We then extend the approach with automatically generated message translators to provide a unified mediation framework [Bennaceur et al. 2015] that deals with interoperability at both the application and middleware layers. However, this mediation framework ensures the correct composition between already selected components (capabilities). The collaborative security framework we present is this paper is requirement driven and therefore is able to select and configure the capabilities automatically. Letier and Heaven [Letier and Heaven 2013] propose to use mediator (controller) synthesis to derive a machine specification that satisfies one requirement and then compose them to form a specification that satisfies a set of requirement. Hence, combining requirement modelling and mediator synthesis help in dealing with multiple requirements of the system. Rather than the synthesis of one specification to satisfy requirements, our approach focuses on the configuration and mediation of the existing behavioural specification. Cavallaro *et al.* [Cavallaro et al. 2012] propose to extend the KAOS goal models in order to define a specifications of services, which are then instantiated at runtime. In this case, mediators are used to compensate for the differences between the discovered service instance and the service specification rather than to select the services and coordinate the associated behaviours.

Existing solutions for the generation of mediators require exploring all possible combination of behaviours. As a result, they can rapidly become prohibitive when dealing with alternative selections of components, and the corresponding behaviours. By first selecting the features to be enabled on a subset of components, then projecting the behaviours of the selected components onto the enabled features, our approach reduces the analysis space for the mediator synthesis. Rodrigues *et al.* [Rodrigues et al. 2015] also assemble components at the architectural level considering the components' interfaces, then analyse the composed behaviour of the selected (bounded) interfaces. Nevertheless, the approach assumes that the components to assemble are interoperable, i.e. they can be bound together and interact correctly without any mediator. Nejati *et al.* [Nejati et al. 2012] combines both structural and behavioural analysis for feature composition. Our approach also reasons about both structural and behavioural properties but the synthesis of mediators *enables* rather than simply *checks* the desirable behavioural properties. Greenyer *et al.* propose to synthesise mediators incrementally using the commonalties of different products (a.k.a. feature configuration) in a product line [Greenyer et al. 2013]. However, they do not consider the selection, let alone the optimisation of the selection, of features to satisfy given requirements.

Using multiple components to develop secure systems has been the subject of a great deal of work, especially at the network level [Meng et al. 2015]. However, these components often have similar capabilities and are designed to collaborate in order to implement security controls. In our approach, the IoT devices are not specifically designed or intended for security purposes. Evidence of the relevance of such a process has been given by the use of some toy robots such as Spykee for home protection [Wayner 2010]. However, as the number, complexity, and heterogeneity of connected devices and people in the IoT increases, the attack surface is widened and uncertain [Covington and Carskadden 2013] and it also becomes more difficult to scope the security problem by specifying the stakeholders involved, the assets and their values, and the potential threats [Haley et al. 2008]. Existing work focuses on securing the interaction between the IoT devices at the network, middleware, and application layers [Sicari et al. 2015], and targeting only information security. But as technology becomes more entwined with the physical world, safeguarding personnel, information, equipment, IT infrastructure, facilities and other material assets become paramount [Cerf 2015]. Our collaborative security framework leverages the capabilities of IoT devices in order to provide adaptive software solution for physical security– delivering security '*by*' the IoT rather than security '*of*' the IoT.

To deal with the inherent mobility of the devices and people as well as the diversity of applications in the IoT, software systems must adapt their structure, behaviour, and security mechanisms [Cheng et al. 2009]. Adaptive security (sometimes called self-protection [Yuan et al. 2014]) aims to enable systems to vary their protection in the face of changes in their operational environment. A requirements-driven approach for adaptive security enables the analysis and reasoning about the cost and benefit of the security controls. Salehie *et al.* [Salehie et al. 2012] propose an approach in which a runtime model that combines goals, threats, and assets models is used to evaluate the cost and benefit of applying each security control and choosing the most appropriate one. The focus of adaptive security has mainly been on the effective selection of security controls according to contextual information. Rather than *what* security controls need to be implemented, our work addresses *how* security controls can be implemented. Furthermore, while techniques for adaptive security assume the security control to be implemented, we assume only a specification of the security control while its implementation is realised at runtime by making the available IoT devices collaborate.

8. CONCLUSIONS & FUTURE WORK

The collaborative security framework described in this paper provides a systematic, tool-supported approach for satisfying security requirements through the composition of multiple capabilities. Our contribution stems from the synergetic use of feature modelling and mediator synthesis and its application to security. Our approach relies on rich, multiple models so that many facets of capabilities and requirements can be captured and analysed. The main advantage of the approach is to satisfy requirements by introducing security controls without the need to rebuild or even deploy additional devices. We implemented a collaborative security framework that computes the optimal set of features to be enabled on a subset of capabilities in order to realise a security control, then generates a mediator that composes the selected capabilities in order to satisfy the behavioural specification of the chosen security control. We used this framework to make multiple robots collaborate in order to protect a mobile phone from theft. We showed that the performance for realising collaboration makes it easily applicable at runtime. Our results provide initial evidence that the IoT can play an important role in enabling security by offering the infrastructure to connect multiple devices on the fly in order to implement adequate security controls.

We also identified several areas for future work. We will investigate the impact of the trustworthiness of individual capabilities on the collaboration. The goal is to ensure the satisfaction of security requirements even when some capabilities are corrupted or compromised. We also aim to relax some of the assumptions of our framework and improve its performance. For example, we are investigating more efficient ways to iterate between the feature selection and the featured-based synthesis by tracing back to the causes of failure in the mediator synthesis and informing the feature selection. We are also planning to explore how to include human agents within the collaborative behaviour. We also plan to deploy our framework in a smart home (or smart city) with a large number of capabilities for a prolonged period of time and see how it scales and how people react to it. We believe that the work on collaborative security is a fertile research area that holds great promise for securing increasingly prevalent cyber-physical systems.

REFERENCES

A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash. 2015. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Comm. Surveys Tutorials* 17, 4 (2015), 2347–2376.

Mikael Asplund and Simin Nadjm-Tehrani. 2016. Attitudes and Perceptions of IoT Security in Critical Societal Services. *IEEE Access* 4 (2016), 2130–2138.

Luigi Atzori, Antonio Iera, and Giacomo Morabito. 2010. The Internet of Things: A survey. *Computer Networks* 54, 15 (2010), 2787–2805.

Christel Baier, Marcus Größer, Martin Leucker, Benedikt Bollig, and Frank Ciesinski. 2004. Controller Synthesis for Probabilistic Systems. In *Proc. of the 3rd Int. Conf. on Theoretical Computer Science, TCS*. 493–506.

Amel Bennaceur, Emil Andriescu, Roberto Speicys Cardoso, and Valérie Issarny. 2015. A Unifying Perspective on Protocol Mediation: Interoperability in the Future Internet. *J. of Internet Services and Applications* 6, 1 (2015), 12:1–12:15.

Amel Bennaceur, Arosha K. Bandara, Michael Jackson, Wei Liu, Lionel Montrieux, Thein Than Tun, Yijun Yu, and Bashar Nuseibeh. 2014. Requirements-driven mediation for collaborative security. In *9th Int. Symp. on Softw. Eng. for Adaptive and Self-Managing Syst., SEAMS*. 37–42.

Amel Bennaceur and Valérie Issarny. 2015. Automated Synthesis of Mediators to Support Component Interoperability. *IEEE Trans. Softw. Eng.* 41, 3 (2015), 221–240.

Patrik Berander and Anneliese Andrews. 2005. Requirements Prioritization. In *Engineering and Managing Software Requirements*, Aybüke Aurum and Claes Wohlin (Eds.). Springer Berlin Heidelberg, 69–94.

Rodney Brooks. 2009. Robots Everywhere!. In *Computing Research That Changed the World: Reflections and Perspectives, CRASS '09*. Article 13, 39 pages.

Javier Cámara, Gabriel A Moreno, and David Garlan. 2015. Reasoning about Human Participation in Self-Adaptive Systems. In *Proc. of the 10th Int. Symp. on Softw. Eng. for Adaptive and Self-Managing Syst., SEAMS*. 146–156.

Luca Cavallaro, Pete Sawyer, Daniel Sykes, Nelly Bencomo, and Valérie Issarny. 2012. Satisfying requirements for pervasive service compositions. In *Proc. of the 7th Workshop on Models@run.time*. 17–22.

Vinton G. Cerf. 2015. Prospects for the Internet of Things. *XRDS* 22, 2 (Dec. 2015), 28–31.

Betty H. C. Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi, and Jeff Magee *et al.* 2009. Software Engineering for Self-Adaptive Systems: A Research Roadmap. In *Softw. Eng. for Self-Adaptive Syst. [outcome of a Dagstuhl Seminar]*. 1–26.

Edmund M. Clarke and Jeannette M. Wing. 1996. Formal Methods: State of the Art and Future Directions. *Comput. Surveys* 28, 4 (1996), 626–643.

Andreas Classen, Quentin Boucher, and Patrick Heymans. 2011. A text-based approach to feature modelling: Syntax and semantics of TVL. *Sci. Comput. Program.* 76, 12 (2011), 1130–1143.

Andreas Classen, Maxime Cordy, Pierre-Yves Schobbens, Patrick Heymans, Axel Legay, and Jean-François Raskin. 2013. Featured Transition Systems: Foundations for Verifying Variability-Intensive Systems and Their Application to LTL Model Checking. *IEEE Trans. Softw. Eng.* 39, 8 (2013), 1069–1089.

M.Journal Covington and R. Carskadden. 2013. Threat implications of the Internet of Things. In *Cyber Conflict (CyCon), 2013 5th Intl. Conf. on*. 1–12.

Nicolás D'Ippolito, Víctor A. Braberman, Nir Piterman, and Sebastián Uchitel. 2013. Synthesizing nonanomalous event-based controllers for liveness goals. *ACM Trans. Softw. Eng. Methodol.* 22, 1 (2013).

Rüdiger Ehlers. 2011. Generalized Rabin(1) Synthesis with Applications to Robust System Synthesis. In *Proc. of NASA Formal Methods - Third Int. Symp., NFM*. 101–115.

Joel Greenyer, Christian Brenner, Maxime Cordy, Patrick Heymans, and Erika Gressi. 2013. Incrementally synthesizing controllers from scenario-based product line specifications. In *Proc. of the Joint Meeting of the Euro. Softw. Eng. Conf. and the Symp. on the Found. of Softw. Eng., ESEC/FSE*. 433–443.

Charles B. Haley, Robin C. Laney, Jonathan D. Moffett, and Bashar Nuseibeh. 2008. Security Requirements Engineering: A Framework for Representation and Analysis. *IEEE Trans. Softw. Eng.* 34, 1 (2008).

HyperCat Consortium. 2016. HyperCat - Home. (2016). http://www.hypercat.io/.

IERC. 2015. *IoT Semantic Interoperability*. IERC-European Research Cluster on the Internet of Things.

IFR Statistical Department. 2014. *World Robotics 2014 - Service Robots*. Technical Report. IFR - Int. Federation of Robotics.

Paola Inverardi and Massimo Tivoli. 2013. Automatic Synthesis of Modular Connectors via Composition of Protocol Mediation Patterns. In *Proc. of the 35th Int. Conf. on Softw. Eng., ICSE*. 3–12.

Michael Jackson and Pamela Zave. 1995. Deriving Specifications from Requirements: An Example. In *Proc. of the 17th Int. Conf. on Softw. Eng., ICSE*. 15–24.

Michael C. Jaeger, Gregor Rojec-Goldmann, and Gero Mühl. 2005. QoS Aggregation in Web Service Compositions. In *Proc. of the IEEE Int. Conf. on e-Technology, e-Commerce, and e-Services, EEE*. 181–185.

Michael Jeronimo and Jack Weast. 2003. *UPnP Design by Example :A Software Designer's Guide to Universal Plug and Play*. Intel Press.

Kyo C Kang, Sholom G Cohen, James A Hess, William E Novak, and A Spencer Peterson. 1990. *Feature-oriented domain analysis (FODA) feasibility study*. Technical Report. DTIC Document.

Richard M. Karp. 1972. Reducibility among combinatorial problems. In *Proc. of a Symp. on the Complexity of Computer Computations*. 85–103.

Robert M. Keller. 1976. Formal Verification of Parallel Programs. *Commun. ACM* 19, 7 (1976), 371–384.

Emmanuel Letier and William Heaven. 2013. Requirements modelling by synthesis of deontic input-output automata. In *35th Int. Conf. on Softw. Eng., ICSE*. 592–601.

Linux Foundation. 2016a. AllJoyn: An open source software framework. (2016). https://allseenalliance.org/framework.

Linux Foundation. 2016b. IoTivity: open source software framework. (2016). https://www.iotivity.org/.

Valerio Panzica La Manna, Joel Greenyer, Carlo Ghezzi, and Christian Brenner. 2013. Formalizing correctness criteria of dynamic updates derived from specification changes. In *Proc. of the 8th Int. Symp. on Softw. Eng. for Adaptive and Self-Managing Syst., SEAMS*. 63–72.

Radu Mateescu, Pascal Poizat, and Gwen Salaün. 2012. Adaptation of Service Protocols Using Process Algebra and On-the-Fly Reduction Techniques. *IEEE Trans. Softw. Eng.* 38, 4 (2012), 755–777.

Guozhu Meng, Yang Liu, Jie Zhang, Alexander Pokluda, and Raouf Boutaba. 2015. Collaborative Security: A Survey and Taxonomy. *ACM Comput. Surv.* 48, 1, Article 1 (July 2015), 42 pages.

Shiva Nejati, Mehrdad Sabetzadeh, Marsha Chechik, Steve M. Easterbrook, and Pamela Zave. 2012. Matching and Merging of Variant Feature Specifications. *IEEE Trans. Softw. Eng.* 38, 6 (2012), 1355–1375.

Amir Pnueli. 1977. The Temporal Logic of Programs. In *Proc. of the 18th Annual Symp. on Foundations of Computer Science*. 46–57.

Amir Pnueli and Roni Rosner. 1989. On the Synthesis of a Reactive Module. In *Proc. of the 16th Annual Symp. on Principles of Programming Languages, POPL*. 179–190.

Pedro Rodrigues, Jeff Kramer, and Emil Lupu. 2015. On Re-Assembling Self-Managed Components. In *Proc of the Int. Symp. on Integrated Network and Service Management, IM*. 727–733.

F. Rossi, P. Van Beek, and T. Walsh. 2006. *Handbook of constraint programming*. Vol. 35. Elsevier Science.

Mazeiar Salehie, Liliana Pasquale, Inah Omoronyia, Raian Ali, and Bashar Nuseibeh. 2012. Requirements-driven adaptive security: Protecting variable assets at runtime. In *Proc. of the 20th IEEE Int. Req. Eng. Conf., RE*. 111–120.

Sabrina Sicari, Alessandra Rizzardi, Luigi Alfredo Grieco, and Alberto Coen-Porisini. 2015. Security, privacy and trust in Internet of Things: The road ahead. *Computer Networks* 76 (2015), 146–164.

Martin R. Stytz. 2004. Considering Defense in Depth for Software Applications. *IEEE Security & Privacy* 2, 1 (2004), 72–75.

Tian Huat Tan, Manman Chen, Jun Sun, Yang Liu, Étienne André, Yinxing Xue, and Jin Song Dong. 2016. Optimizing selection of competing services with probabilistic hierarchical refinement. In *Proc. of the 38th Int. Conf. on Softw. Eng., ICSE*. 85–95.

Roman Vaculín, Roman Neruda, and Katia P. Sycara. 2009. The process mediation framework for semantic web services. *Int. J. of Agent-Oriented Softw. Eng., IJAOSE* 3, 1 (2009), 27–58.

Rob van der Meulen and Janessa Rivera. 2014. *Gartner Says a Typical Family Home Could Contain More Than 500 Smart Devices by 2022*. Technical Report. Gartner. http://www.gartner.com/newsroom/id/2839717

Axel van Lamsweerde. 2009. *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley.

Peter Wayner. 2010. Protecting Your Home From Afar With a Robot. The New York Times, Online. (November 2010). http://www.nytimes.com/2010/11/04/technology/personaltech/04basics.html

Steve H Weingart. 2000. Physical security devices for computer subsystems: A survey of attacks and defenses. In *Int. Workshop on Cryptographic Hardware and Embedded Syst.* Springer, 302–317.

Gio Wiederhold. 1992. Mediators in the Architecture of Future Information Systems. *IEEE Computer* 25, 3 (1992), 38–49.

Daniel M. Yellin and Robert E. Strom. 1997. Protocol Specifications and Component Adaptors. *ACM Trans. on Programming Languages and System, TOPLAS* 19, 2 (1997), 292–333.

Yijun Yu, Virginia N. L. Franqueira, Thein Than Tun, Roel Wieringa, and Bashar Nuseibeh. 2015. Automated analysis of security requirements through risk-based argumentation. *J. of Syst. and Softw.* 106 (2015), 102–116.

Eric Yuan, Naeem Esfahani, and Sam Malek. 2014. A Systematic Survey of Self-Protecting Software Systems. *ACM Trans. on Autonomous and Adaptive Syst., TAAS* 8, 4 (2014), 17.

Liangzhao Zeng, Boualem Benatallah, Anne H. H. Ngu, Marlon Dumas, Jayant Kalagnanam, and Henry Chang. 2004. QoS-Aware Middleware for Web Services Composition. *IEEE Trans. Softw. Eng.* 30, 5 (2004), 311–327.