Engineering Topology Aware Adaptive Security: Preventing Requirements Violations at Runtime

Christos Tsigkanos*, Liliana Pasquale†, Claudio Menghi*, Carlo Ghezzi*, and Bashar Nuseibeh†‡
* Politecnico di Milano, Milano, Italy
† Lero - The Irish Software Engineering Research Centre, Limerick, Ireland
‡ Department of Computing, The Open University, Milton Keynes, UK

***Abstract*—Adaptive security systems aim to protect critical assets in the face of changes in their operational environment. We have argued that incorporating an explicit representation of the environment's topology enables reasoning on the location of assets being protected and the proximity of potentially harmful agents. This paper proposes to engineer topology aware adaptive security systems by identifying violations of security requirements that may be caused by topological changes, and selecting a set of security controls that prevent such violations. Our approach focuses on physical topologies; it maintains at runtime a live representation of the topology which is updated when assets or agents move, or when the structure of the physical space is altered. When the topology changes, we look ahead at a subset of the future system states. These states are reachable when the agents move within the physical space. If security requirements can be violated in future system states, a configuration of security controls is proactively applied to prevent the system from reaching those states. Thus, the system continuously adapts to topological stimuli, while maintaining requirements satisfaction. Security requirements are formally expressed using a propositional temporal logic, encoding spatial properties in Computation Tree Logic (CTL). The Ambient Calculus is used to represent the topology of the operational environment - including location of assets and agents - as well as to identify future system states that are reachable from the current one. The approach is demonstrated and evaluated using a substantive example concerned with physical access control.**

I. Introduction

Adaptive security systems aim to protect critical assets in the face of changes in their operational environment. They do so by monitoring and analysing this environment and deploying security controls that satisfy some security requirements. A key characteristic for engineering adaptive security is the *topology* of the operational environment [20] that can denote the structure of a physical space, such as a building, the location of assets and agents in that space and their structural relationships. These relationships may determine whether assets or agents are co-located (proximity), if an agent can access a specific asset or location (reachability), or if an asset, agent or an area is enclosed by another area (containment).

In addition to existing context models [1], topology can provide a system with both structural and semantic awareness of important contextual characteristics that can affect security concerns. Security requirements can be expressed in terms of proximity and reachability relationships among assets and agents; for example, a security requirement can specify that an asset should never be co-located next to an unauthorised agent who can harm its integrity. The location of agents, who can harm the assets placed in their vicinity raises potential security threats. For example, a threat can arise if a malicious agent can reach a valuable asset from the area in which she is located. Knowing where valuable assets are placed and their relationships to other objects in their proximity is also important in order to identify possible security controls that can be enacted to protect them. For example, authorisation mechanisms may be needed in some of the areas that could be accessed by a malicious agent to harm an asset.

Changes in topology due to movements of assets or agents, or due to changes in the structure of a space can affect system security concerns and determine violations of security requirements. For example, the movement of a valuable asset to a different location may lead to a violation of the asset's confidentiality because illegitimate users can reach the new asset's location. Movements of agents and changes in the structure of a space can also bring new threats. In particular, agents' movements may cause harm to the assets that can be reached from their current location. Moreover, merging two adjacent rooms can make a valuable asset located in one of the rooms reachable by potentially malicious agents that can access the other room.

This paper proposes to engineer adaptive security systems by identifying and preventing violations of security requirements triggered by changes in the topology of the operational environment. To achieve this aim, a live representation of the topology is maintained at runtime and is updated when assets are moved, agents perform actions, or the structure of the space is altered. This allows reasoning on the consequences that topological changes can have on the satisfaction of security requirements at runtime. When the topology changes, we look ahead at a subset of the future system states; these states represent the topological configurations that are reachable when agents perform a sequence of actions. If a security requirement can be violated in one of the future system states, a set of security controls is applied, by revoking from some agents the rights to perform certain actions that lead the system to undesired topological configurations. Security controls are selected by using different criteria, such as whether they satisfy other non-security requirements or minimise the cost (i.e. minimise the number of security controls applied).

We acknowledge SFI grant 10/CE/I1855 and ERC Adavanced Grants (ASAP) no. 291652 and (SMScom) no. 227977.

978-1-4799-3033-3/14 © 2014 IEEE

RE 2014, Karlskrona, Sweden

This paper focuses on topologies describing the structure of a physical area or a building, in contrast to digital topologies which have been the subject of discussion elsewhere [20]. We propose a systematic model-based software engineering approach to formally reason on system properties and enforce security requirements satisfaction. The Ambient Calculus [6] is used to represent the topology of the operational environment - including location of assets and agents - as well as to identify future topological configurations that are reachable from the current one.

Requirements, expressing desired properties related to access control, are formally expressed using Computation Tree Logic (CTL) [9], where spatial properties are encoded by using a set of atomic propositions. The proposed approach is illustrated and evaluated through a substantive example concerned with physical access control. Our experimental results demonstrate the effectiveness of the approach in selecting security controls that avoid entering topological configurations where security requirements are violated. The number of system states to be analysed at runtime is also greatly reduced by excluding agents' actions and locations that do not influence the satisfaction of security requirements.

The rest of the paper is structured as follows. Section II describes the case study adopted throughout the paper to explain and evaluate our work, and Section III provides an overview of our approach. Section IV introduces the formalisms adopted to represent the topology of the operational environment, the system state space and the system requirements. Section V explains the threat analysis performed to detect potential violations of security requirements. Section VI describes the approach adopted to identify and select security controls. Section VII provides experimental results, Section VIII describes related work, and Section IX concludes the paper.

II. Case Study

As a case study we consider a system regulating access to a building hosting lecture theaters and staff offices. Figure 1 represents the map of the building floor layout in which access to locations (rooms and areas) must be regulated. The building is composed of an entrance area (Bld) and the offices areas (A1 and A2). From the building entrance (Bld) one can access lecture theaters LT1 and LT2, as well as the non-faculty offices area (A1). From A1 it is possible to access the researchers' offices (O1, O2, and O4), the printer room (O3), and the faculty's offices area (A2). From A2, it is possible to access professors' offices (O5 and O7), and a room were a safe is located (O6).

In our scenario, valuable assets need to be protected, such as the safe's security code and a new server that will be positioned in office O2. Agents roaming in the building can for example be postdocs or professors, or external entities such as visiting technicians. The access control system should restrict access to the various areas of the building to pursue the following requirements, which here are informally expressed in terms of structural relationships among assets and agents in the physical space.

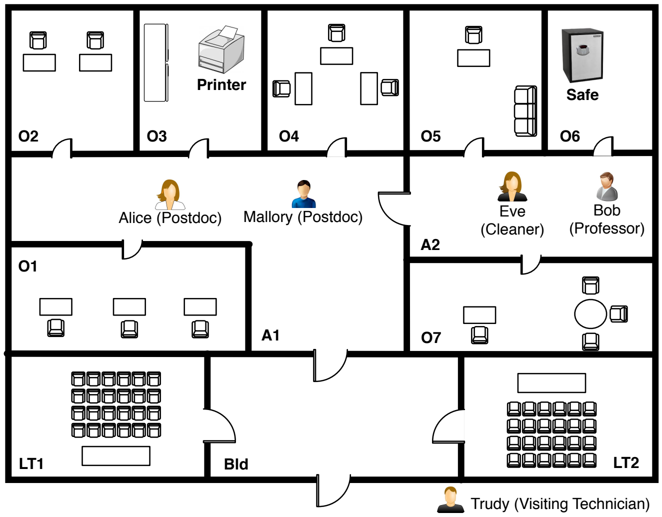


Fig. 1. Map of the academic building used in our case study.

Security Requirements aim to protect valuable assets in the building from damage, theft or improper use. Specifically,

- *SR1*: To guarantee the confidentiality of the safe's security code, this security requirement asserts that no one can be co-located in the safe room (O6) together with one of the agents (e.g., Bob) who knows the safe's security code.
- *SR2*: To preserve the integrity of the server, this security requirement states that only when authorised staff are present (e.g., Alice), other people can be in the same room with the server.
- *SR3*: To prevent harm to persons or assets located in the building, this security requirement claims that an external visitor must always be in specific public areas or in the room where she has to carry out her work. For example, since the visiting technician (Trudy) has to fix the printer located in O3, she can only be in Bld, A1, or O3.

Functional Requirements aim to guarantee that the research centre staff can perform their own work. In particular,

- *FR1*: Alice must be able to access her office O2.
- *FR2*: Bob must be able to access the room O6, which contains the safe.
- *FR3*: The visiting technician must be able to access the room O3, where the printer to be repaired is located.

We envisage different scenarios characterised by topological changes that might require adapting existing authorisation permissions.

Scenario 1. Agents' movements can change the security controls that need to be applied. For example, if a professor (Bob) enters in the safe room, no one else should be allowed to access O6 until the professor exits, in order to satisfy security requirement *SR1*. Similarly, if the cleaner (Eve) is in O6, authorisation to access O6 should be revoked from Bob until Eve exits, in order to guarantee the safe code's confidentiality.

Scenario 2. A new asset (server) is placed in O2 and introduces security requirement *SR2* aiming to guarantee the server's integrity. In this scenario, when a potentially malicious, unauthorised agent (e.g., Mallory), enters the building authorisation mechanisms must be adapted in order to avoid Mallory being co-located with the server, without the presence of an authorised person (e.g., Alice).

Scenario 3. A technician (Trudy) visits the building to fix the printer and therefore security requirement *SR3* must also be satisfied. In this case, Trudy should be authorised to enter only the areas that have to be traversed to reach O3.

III. Topology Aware Adaptive Security

Topology is the study of shapes and spaces, including properties such as connectedness and boundary. We define topology aware adaptive security as the *adaptation process that aims to continue to satisfy security requirements at runtime, even when the structure of the operational environment changes*. Figure 2 depicts the activities of the MAPE (Monitoring, Analysis, Planning, Execution) loop [13] supported by our approach in order to engineer topology aware adaptive security. These activities rely on a representation of the topology, which is kept in sync with the structure of the physical space and the location of assets and agents in that space. The system security and non-security requirements (respectively S and NS in Figure 2) are also modelled explicitly in order to configure the analysis and planning activities.

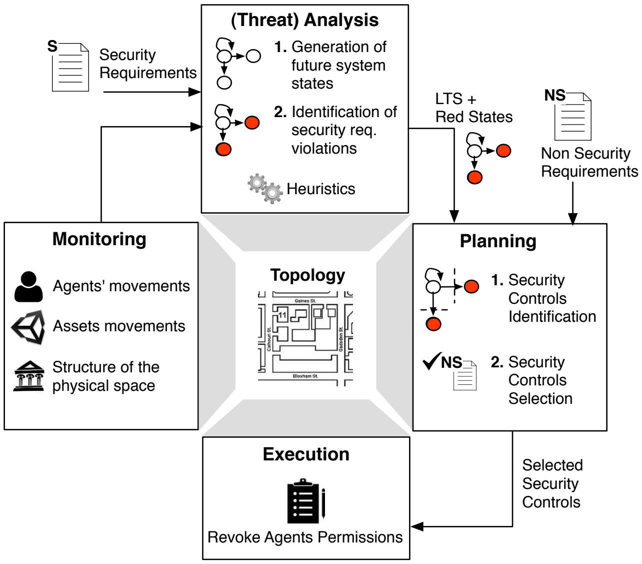

(Threat) Analysis
1. Generation of future system states
2. Identification of security req. violations
Heuristics
S
Security Requirements
LTS + Red States
NS
Non Security Requirements
Monitoring
Agents' movements
Assets movements
Structure of the physical space
Topology
Gaines St.
11
Calhoun St.
Gadsden St.
Bloxham St.
Planning
1. Security Controls Identification
✓NS
2. Security Controls Selection
Execution
Revoke Agents Permissions
Selected Security Controls

Fig. 2. The MAPE loop to support topology aware adaptive security.

Every time a change in the topology of the operational environment or in the system requirements is observed, a new adaptation cycle through the MAPE loop is triggered. This process is agnostic of previous adaptation cycles and security controls are selected independently each time.

Monitoring captures topological changes that can be determined by agents' movements, assets movements, and alterations of the structure of a physical space, and updates the model of the topology accordingly. Note that assets and agents' movements can be monitored automatically by using, for example, smart cards. Alterations to the physical space, on the other hand, require the manual intervention of a human agent to be included in the representation of the topology.

Analysis is triggered by topology changes detected during the monitoring activity and discovers potential threats that may cause requirements violations. To achieve this aim, future systems states are generated and violations of security requirements are identified. The generation of future system states is based on the look-ahead of subsequent executions of actions (i.e. movements) by the agents within the physical space. Assets movements and modifications in the structure of the physical space are monitored as possible exogenous topological changes that may occur. The threat analysis checks that endogenous actions by agents do not lead to violations of security requirements. States in which a violation takes place are also identified (dark states in Figure 2). We also propose a set of heuristics to reduce the system state space by not considering parts of the topology which are not relevant for the satisfaction of security requirements. Threat analysis can also be employed at design time, when it is more feasible to undertake an exhaustive look-ahead of all possible sequences of actions that agents can perform. However, applying threat analysis only at design time reduces effectiveness of detecting unexpected security threats determined by changes in the topology or in the security requirements arising at runtime.

Planning identifies security controls by detecting a suitable set of actions that have to be forbidden to specific agents in order to avoid security requirements violations. The configuration of security controls that is selected should also satisfy other non security requirements, if possible.

Execution changes existing system authorisations by revoking from agents the right to perform the actions forbidden by the selected configuration of security controls. However, other security controls [10] (e.g., obligation and dispensation) can also be supported.

IV. Modelling Formalisms

This section introduces the Ambient Calculus, Labelled Transition Systems (LTS) and Computation Tree Logic (CTL), which are the formalisms adopted to model the topology of the operational environment, the system state space, and the system requirements, respectively.

A. Ambient Calculus

The Ambient Calculus is a process algebra having a special focus on mobility [6]. An *ambient* is an abstract entity that can model different elements both in a physical space (e.g., agents and locations) and in a digital space (e.g., programming scopes and variables) [17]. Ambients reside in a hierarchy of locations and form a tree structure that can be dynamically re-configured when they exercise a set of capabilities (actions), such as *in*, *out*, and *open*. In this work, a fragment of the Ambient Calculus is considered where the communication primitives and the open capability are neglected.

The syntax of the Ambient Calculus used in this paper is described in formulae 1a to 1f, overleaf. A process P

can simply do nothing (formula 1b), can be decomposed in two processes running in parallel (formula 1c), or can be enclosed into an ambient which is a particular kind of process (formula 1d). A process can also execute a capability and then proceed to the execution of another process. We assume that the capabilities available are in and out (formulae 1e and 1f, respectively).

$$\begin{array}{llll}
P, Q, R & ::= & \mathit{processes} & \text{(1a)} \\
 & \mid 0 & \mathit{inactivity} & \text{(1b)} \\
 & \mid P \mid Q & \mathit{parallel\ composition} & \text{(1c)} \\
 & \mid n[P] & \mathit{ambient} & \text{(1d)} \\
 & \mid \mathit{in}\ n.P & \mathit{capability\ to\ enter}\ n & \text{(1e)} \\
 & \mid \mathit{out}\ n.P & \mathit{capability\ to\ exit}\ n & \text{(1f)}
\end{array}$$

An Ambient Calculus formula can represent both the description of the structure of an environment and its evolution. The former expresses how ambients are structured and nested, while the latter describes how the structure of the environment can evolve through the execution of a given set of capabilities.

We represent the topological configuration of the operational environment using Ambient Calculus formulae. The hierarchical relation is exploited to describe how different ambients are nested. We explicitly distinguish among different types of processes representing assets, locations, and agents. For example, the topology of the building described in Section II is encoded as Ambient Calculus formulae 2a-2c[1]. Formula 2a denotes that locations LT1, LT2 and A1 are on the same hierarchical level, since to access them an agent must be located in the building entrance (Bld). These formulae also specify the locations of the agents, namely that Trudy is inside Bld, Alice and Mallory are in A1, Eve and Bob are in A2, and the Server and the Safe are located in O2 and O6, respectively.

$$\begin{array}{ll}
Bld[LT1 \mid LT2 \mid A1 \mid Trudy] & \text{(2a)} \\
A1[Alice \mid Mallory \mid O1 \mid O2[Server] \mid O3 \mid O4 \mid A2] & \text{(2b)} \\
A2[Eve \mid Bob \mid O5 \mid O6[Safe] \mid O7] & \text{(2c)}
\end{array}$$

We assume that assets and locations cannot perform any capability, while agents can always execute any possible capability depending on their current location. More precisely, an agent can always enter (in) all co-located areas and exit (out) from the current room or area. For example, in the configuration described in formulae 2a-2c, Trudy can perform $out\ Bld$, $in\ LT1$, $in\ A1$ or $in\ LT2$, while Bob and Eve can perform $out\ A2$, $in\ O5$, $in\ O6$, and $in\ O7$. Therefore, we do not explicitly represent capabilities into an Ambient Calculus formula, as they can be inferred from how the ambients are structured and from the types of ambients in the topology.

B. Labelled Transition Systems

Labelled Transition System [9] (LTS) is a modelling formalism used to describe systems and their evolution in terms of states and transitions. States usually specify the possible configurations of the system. Transitions describe how the configuration of the system can change by moving from one state to its successors.

Formally, given a set AP of atomic propositions, a LTS $\mathcal{M}$ is formally described as a tuple $\mathcal{M} = \langle S, S_0, R, L \rangle$, where:

- S is a finite set of states;
- $S_0 \subseteq S$ is the set of initial states;
- $R \subseteq S \times S$ is a transition relation that must be *total*, that is for every state $s \in S$ there is a state $s' \in S$ such that $(s, s') \in R$;
- $L : S \rightarrow 2^{AP}$ is a function that labels each state with the set of atomic propositions that are true in that state.

C. From Ambient Calculus to LTS

Starting from an Ambient Calculus formula, interpreting it over an LTS means describing the topological evolution of the system based on execution of capabilities. Each LTS state represents a different topological configuration. Atomic propositions are used to describe where agents and assets are located (spatial modalities). For example, atomic proposition '$Bob\ in\ O6$' is true when Bob is located in office O6. LTS transitions connect different states and correspond to the execution of capabilities by agents.

D. Computation Tree Logic

CTL [9] is a branching temporal logic used to specify temporal properties that a system must satisfy. CTL includes two types of formulae: *state* and *path* formulae. State formulae are defined over a set of atomic propositions AP using the grammar specified in 3, where $a \in AP$ and φ is a path formula. A state formula can be an atomic proposition a, the special proposition $true$, the composition ($\wedge$) of two sub-formulae, the negation ($\neg$) of a formula, and a path CTL formula prefixed by E (exists) or A (always) path quantifiers. E predicates that φ must hold on at least one path starting from the current state, while A asserts that φ must hold on all paths starting from the current state.

$$\Phi ::= \mathit{true} \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid E\varphi \mid A\varphi \tag{3}$$

CTL *path formulae* are defined in 4. A state CTL formula Φ prefixed by the *next* operator (X), and two CTL formulae Φ_1 and Φ_2 linked by the *until* operator (U) are valid CTL path formulae.

$$\varphi ::= X\Phi \mid \Phi_1 U \Phi_2 \tag{4}$$

A state formula Φ is evaluated over a state s of the LTS. For example, property $\Phi = \Phi_1 \wedge \Phi_2$ is true in a state s if and only if s satisfies both Φ_1 and Φ_2. Similarly, property $E\varphi$ is true in a state s of the LTS iff there exists a path π starting from s that satisfies φ. Path formulae are interpreted over infinite paths of the LTS. For example, given an infinite path π, the property $\Phi_1 U \Phi_2$ is true if there exists a state s in the path that satisfies Φ_2 and each state that precedes s on the path satisfies Φ_2. The interested reader can refer to [9] for a complete description of the semantics of CTL.

In this paper CTL is used to specify security as well as other requirements. For example, Formulae 5a-5c represent

[1] Note that LT1, ..., O7 is used as a shortcut to $LT1[0]$, ..., $O7[0]$.

security requirements *SR1-SR3*. Variables Y and Z are used to denote (implicitly universally quantified) agents (such as Alice, Mallory, ...) and locations (such as Bld, A1, ...). Moreover, an atomic proposition, such as '$Bob \mid Eve$' denotes that Bob is co-located with Eve.

$$SR1: \ AG(\neg((Bob \mid Y) \wedge (Bob\ in\ O6))) \tag{5a}$$
$$SR2: \ AG(\neg((Y \mid Server) \wedge \neg(Alice \mid Server)) \tag{5b}$$
$$SR3: \ AG(\neg(Trudy\ in\ Z)) \tag{5c}$$

Formulae 6a-6d represent security requirement $SR1$ by associating Y with Alice, Mallory, Eve, and Trudy.

$$AG(\neg((Bob \ \mid \ Alice) \wedge (Bob\ in\ O6)))\wedge \tag{6a}$$
$$AG(\neg((Bob \ \mid \ Mallory) \wedge (Bob\ in\ O6)))\wedge \tag{6b}$$
$$AG(\neg((Bob \ \mid \ Eve) \wedge (Bob\ in\ O6)))\wedge \tag{6c}$$
$$AG(\neg((Bob \ \mid \ Trudy) \wedge (Bob\ in\ O6))) \tag{6d}$$

In this paper CTL is also adopted to express the functional requirements of the case study. In particular, formulae (7a)-(7c) express functional requirements *FR1-FR3*.

$$FR1: \ EF(Alice\ in\ O2) \tag{7a}$$
$$FR2: \ EF(Bob\ in\ O6) \tag{7b}$$
$$FR3: \ EF(Trudy\ in\ O3) \tag{7c}$$

V. Threat Analysis

Threat analysis aims to predict potential violations of security requirements that can take place in future system states. This section describes the process adopted to generate future system states reachable from the current topology. It also explains how violations of security requirements - that can take place in future states - are detected. The section concludes by illustrating a set of heuristics that can be adopted to reduce the state space analysed.

A. Generation of Future System States

Future system states are generated after changes in the topological configuration of the operational environment take place. Topological changes are continuously monitored and the representation of the topology is updated at runtime accordingly. Any subsequent execution of actions by the agents within the physical space (agents' intentions) is exhaustively considered and the corresponding future system states are generated. This is performed because future agents' actions cannot be predicted in advance. Movements of assets and modifications in the structure of the physical space are not considered, because they are conceived as possible exogenous topological stimuli, and therefore cannot determine requirements violations spontaneously.

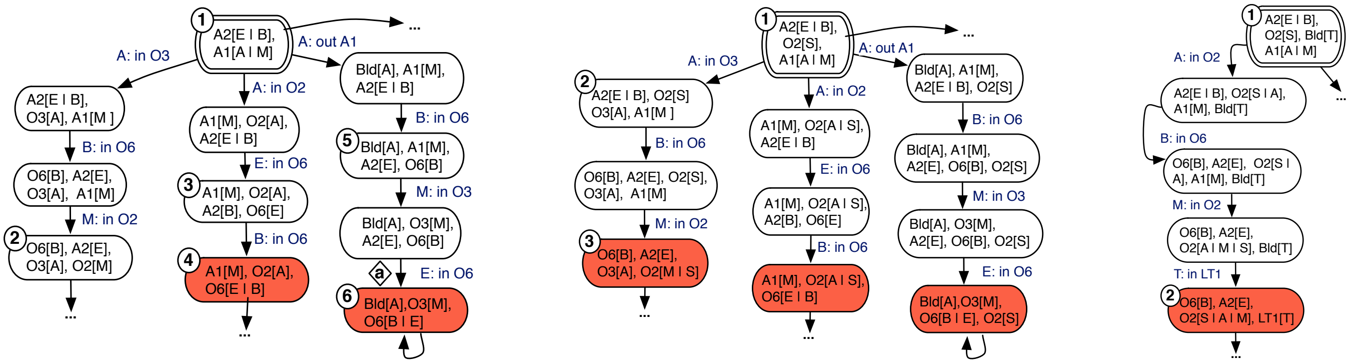
Future system states are generated in two distinct steps: (I) exhaustive look-ahead of action executions by agents and (II) LTS generation, which translates the representation of the topological configuration into a formalism that is suitable for verification. Exhaustive (limited scope) look-ahead of actions execution depends on the structure of the physical space and on the current position of agents. Each agent can enter accessible areas with which she is co-located, and can exit from the area where she is currently placed. However, the exhaustive look-ahead is bounded by the number of steps selected to be considered in the future by the agents. The portion of the state space that is generated has a limited depth. This corresponds to the execution of a predefined number of actions per agent.

We assume a look-ahead bound of one step[2] (i.e. each agent always performs at most one action per location). In this way, at one step in the future, capabilities that can be performed with respect to all locations accessible directly by agents are examined. For example, in Figure 1, Trudy can perform *in LT1*, *in LT2*, *in A1*, and *out Bld*. Each action performed by an agent at a specific step is considered to be performed in parallel with those of the other agents at the same step, in a way that is consistent with process calculi. Any possible ordering in which capabilities can be executed by the agents is also considered. Even though different sequences of the same capabilities can lead to the same topological configuration, their ordering is relevant because it can affect the satisfaction of security requirements differently. More precisely, the generation of all the possible action interleavings is fundamental to identify a specific sequence of actions that lead to the violation of a security requirement. For example, if both Mallory and Alice enter O2 at the same time, the sequence of actions where Mallory accesses O2 before Alice will determine a violation of security requirement *SR2*. In contrast, the same requirement holds if Alice enters O2 before Mallory.

Algorithm 1 sketches the procedure we use to look ahead at capability executions and to generate all the sequences of actions that agents (A_g) can perform from the current topology T_c. For each agent a (Line 2), the set of relevant movement intentions (Line 3) corresponding to accessing (*in*) co-located locations or exiting (*out*) from the current location are considered. Function f associates each agent with co-located rooms and areas. For each of these locations (Line 4), function GETACTION() returns the relevant action that an agent can perform on it (Line 5). The set A_{act} (Line 6) is populated with all the possible capabilities an agent can do depending on the accessible locations ($\{\langle a, action, s\rangle\}$). Function z associates each agent with the possible actions she can execute (Line 8). When multiple agents are present, the product of their potential capabilities is computed (Line 10), to account for different orderings of capability executions. This is equivalent to inserting sets of capabilities into an Ambient Calculus formula representing the current topological configuration. Afterwards, given the current topology T_c, each possible set of interleaving actions is mapped to the transitions of the LTS (Line 11).

To generate the LTS, a set of states is subsequently identified starting from the current one representing the current topology. The procedure used to accomplish this task is the one described by Mardare et al. [16], [15], which uses sets

[2] One might also consider larger scopes, although the larger the scope the more likely it is that exogenous changes happen and invalidate the analysis.



(a) Scenario 1 - Initial Configuration

(b) Scenario 2

(c) Scenario 3

Fig. 3. Fraction of the LTS generated from the case study; violating states are in dark.

Algorithm 1 Exhaustive look-ahead of capability executions

```
function LOOK-AHEAD($T_c$, $A_g$)
    for $a \in A_g$ do
        $A_{act} = \{\}$;
        for $s \in f(a)$ do
            $action$ = GETACTION($a$,$s$);
            $A_{act} = A_{act} \cup \{\langle a, action, s\rangle\}$
        end for
        $z(a) = A_{act}$
    end for
    for $ac \in$ PRODUCT($z(\cdot)$)) do
        MAPONLTS($ac$,$T_c$);
    end for
end function
```

for the representation of state information. Each transition corresponds to the execution of a capability by an agent, while each state of the LTS describes a different topological configuration of the operational environment; note that different states can also be associated with the same topology in case this is reached by performing different sequences of agents' capabilities. Figure 3 partially represents the LTS generated from different topological configurations identified in the scenarios proposed in Section II.

B. Identification of Security Requirements Violations

The LTS representing agents' potential movements is explored to identify those states where security requirements can be violated. Such states are detected by checking the LTS against CTL formulae representing security requirements. Figure 3 highlights in dark the states where security requirements are violated[3] in our scenarios. The attack scenarios appearing in a possible evolution of the system can be summarised as follows.

- In the first scenario (Figure 3a), Trudy is not in the building and the server is not placed in O2 (state 1). The first branch (from the left) of the LTS does not cause any violation of security requirements because although Mallory enters O2 the server is still not placed in it. The action sequences represented by the second and the third branch cause violation of security requirement *SR1*. In the second branch, Eve accesses O6 (state 3) and afterwards Bob accesses the same room (state 4). In the third branch, Bob accesses the safe room (state 5) and subsequently Eve accesses the same room (state 6).
- In the second scenario (Figure 3b), the server (S) is placed in room O2 (state 1). The action sequence represented by the first branch from the left violates security requirement *SR2*. This is because Mallory accesses O2 (state 3), where the server is located, when Alice is not present, since she is in O3 (state 2).
- In the third scenario (Figure 3c), Trudy accesses the building (state 1). An action sequence that may cause a violation of security requirements is when Trudy moves to rooms (e.g., LT1) she is not supposed to traverse to reach the printer room (state 2). This action sequence violates requirement *SR3*.

The theoretical space complexity to identify security requirements violations depends on the number of states and transitions that are generated in the worst case. To compute the number of transitions of the LTS it is necessary to remember that each of them is associated with an action. Given a set of agents A_g and locations L, in the worst case an agent $a_1 \in A_g$, can perform $|L|$ actions that is entering or exiting every location $l \in L$. Since each action performed by a_1 can be combined with every other action performed by another agent, these actions can be aggregated in $|L|^{|A_g|}$ sets of $|A_g|$ actions (one for each agent). It is also necessary to consider the possible permutations of the agents' actions (i.e. all the possible orderings in which actions can be performed). More precisely, a set of $|A|$ actions can be ordered in $|A|!$ ways. Thus, the total number of transitions in the worst case scenario is $|A_g|! \cdot |A_g| \cdot |L|^{|A_g|}$. Since each transition moves the system to a new state, in the worst case scenario the total number of states of the LTS is $|A_g|! \cdot |A_g| \cdot |L|^{|A_g|} + 1$, that is the number of transitions plus the initial state representing the initial topological configuration of the system.

Once a LTS is generated (e.g., $\mathcal{M}$), the complexity of verifying if a formula φ holds in $\mathcal{M}$ is $\mathcal{O}((|S|+|R|) \cdot |\varphi|)$ [9].

[3] Self-loops on final states are added to comply with the LTS definition.

Thus, in our case the time complexity is $\mathcal{O}((|A_g|! \cdot |A_g| \cdot |L|^{|A_g|}) \cdot |\varphi|))$.

C. State Space Reduction Heuristics

State space reduction heuristics aim to reduce the sequences of actions that are considered, and consequently, the number of LTS states generated. This paper proposes two types of domain-specific techniques to achieve such reduction. Firstly, we use the topology of the operational environment to avoid considering agents' actions that are irrelevant to the satisfaction of security requirements. Secondly, a new type of analysis that is reactive instead of proactive is enacted, when other measures that aim to meet runtime demands are ineffective.

The first heuristic depends on the pattern adopted for the specification of security requirements, which indicates how agents, assets and locations are related to each other. In particular, this heuristic only focuses on the agents' actions that can determine the violation of the specified security requirements. Let us consider security requirements that are expressed as $AG(\neg(a_g \ in \ l))$, whose violation is determined in those states where agent a_g reaches location l. The look-ahead of the actions performed by agent a_g can therefore focus only on those allowing a_g to reach location l within the number of steps in the future that are analysed. For example, to check security requirement *SR3* assuming the topological configuration where Trudy is in Bld, it is necessary to consider only the actions that allow Trudy to reach accessible locations $l \in Y$, such as *Trudy in LT1* and *Trudy in LT2*.

Consider security requirements that are expressed as $AG(\neg((a_{g1} \ | \ a_{g2}) \wedge (a_{g2} \ in \ l)))$, whose violation is determined when two agents (a_{g1} and a_{g2}) are co-located in the same place (l). In this case, the look-ahead can focus only on those actions allowing a_{g1} and a_{g2} to reach location l in one step. For example, to check security requirement *SR2*, assuming the topological configuration shown in Figure 1, look-ahead of agents' actions will only include *Bob in O6* and *Eve in O6*.

Consider security requirements that are expressed as $AG(\neg((a_{g1} \mid a_s) \wedge \neg(a_{g2} \mid a_s)))$, whose violation is determined when an unathorised agent (a_{g1}) is co-located with an asset (a_s), while an authorised agent (a_{g2}) is not present. In this case, the look-ahead of the actions performed by a_{g1} will focus only on those that allow the agent to reach the place where the asset is located, while the look-ahead of the actions performed by a_{g2} will focus only on those allowing her to exit the location where the asset is placed or to enter the areas where the asset is not located. For example, to check security requirement *SR2*, assuming the topological configuration shown in Figure 1 when the server is in O2, look-ahead of agents' actions will only include Mallory and Alice entering O2. Configurations where Mallory and Alice enter rooms different than O2 are not taken into account. Note that this heuristic alters function f adopted in Algorithm 1, which specifies the locations on which an agent can exercise a capability, by considering only those locations containing assets and agents that are relevant for the satisfaction of security requirements.

Another technique that can be employed is related to the threat analysis strategy. To counter the complexity explosion that can occur in certain configurations, the system might opt for a reactive mode of operation. Since the theoretical number of LTS states can be maximal in particular circumstances, the proactive approach presented previously might not be effective *at runtime*. Informally, this is likely to occur when the topological configuration requires the look-ahead of interaction of multiple agents with several parallel locations. In such a case, if a likely state explosion is observed in a part of the topology a certain change in operation can be performed to mitigate it.

To handle these situations the system has to switch to a reactive operation mode. More precisely, agents' actions are not considered and by default any action is assumed to determine a threat. Therefore, all locations are considered locked and agents have to request access in order to be able to perform any movement. This way, security requirements verification is performed when agents are attempting to move (e.g., entering co-located areas or exiting from their current place). In this case, the LTS is constructed only with respect to the requested capability execution. Our approach decides to switch to this reactive mode in case the state space exceeds a certain pre-determined threshold. However, the reactive mode is not always sustainable as it can compromise the system's usability by requiring each user to request access on every attempt to change location.

VI. Topology Aware Planning

This section illustrates the algorithm adopted to identify alternative configurations of security controls able to prevent potential violations of security requirements. A candidate configuration is selected among those that also satisfy relevant functional requirements.

A. Security Controls Identification

Algorithm 2 uses the results obtained during the threat analysis to identify alternative configurations of security controls. This recursive algorithm prunes the LTS by progressively removing states where security requirements are violated. This process essentially removes future paths of execution that should never be reachable from the current topology. R, S and S_v indicate the transition relation of the LTS, its states, and the set of states that violate the requirements, respectively. Note that each state s of the LTS that is generated during the threat analysis has exactly one predecessor. Signature $R(s)^{-1}$ returns the predecessor of state s. Recall that each LTS transition represents the execution of a capability by an agent on a specific location (e.g., $\langle Eve, in, O6 \rangle$ corresponds to transition *a* in Figure 3a). However, a similar transition might occur in other parts of the LTS (e.g., the one entering in state 3). We refer to such transitions that refer to the same agent, action, and location, but occur on different parts of the LTS as *homologous*. For each transition entering the violating states (S_v), all its homologous transitions (including itself) are removed, including their successor states. For example, for transition a in Figure 3a, states 3, 4, and 6 are removed.

Algorithm 2 Identification of Security Controls

```
 1: function IdSecurityControls($R$, $S_v$, $S$)
 2:     $sc = \{\{\}\}$ ;
 3:     for $s \in S_v$ do
 4:         $S'_v = S_v \setminus s$;
 5:         $S' = S \setminus s$;
 6:         $s' = R^{-1}(s)$
 7:         $c = cap(s', s)$;
 8:         $\langle S'_v, S, R' \rangle$ = Prune($S'_v$, $R$, $S$, $c$)
 9:         if $S'_v \cap S = \emptyset$ then
10:             $sc = sc \cup \{c\}$ ;
11:         else
12:             $sc'$ = IdSecurityControls($R'$, $S'_v$, $S'$);
13:             for $sc'' \in sc'$ do
14:                 $sc = sc \cup (sc'' \cup \{c\})$;
15:             end for
16:         end if
17:     end for
18:     return $sc$;
19: end function
```

The set sc contains the configuration of security controls that is computed in the current iteration of the IdSecurityControls procedure. First, the algorithm creates sc (Line 2). If there are no states that violate the security requirements, the empty set is returned (Line 18), as no security controls should be applied. Each state s that violates a security requirement (Line 3) is iteratively removed from the set of violating states (Line 4) and from the set of states of the LTS (Line 5). To make this state not reachable, its predecessor s' (Line 6) is analyzed and the capability c that labels the transition from s' to s is identified (Line 7). The graph is pruned by removing all the transitions that are homologous to c (Line 8). This represents the effect of the security control associated with c, which revokes from an agent the permission to perform the action represented by c. If the new set of violating states (S'_v) does not contain any other state (Line 9), it means that forbidding the capability c (alone) allows the satisfaction of the security requirement, and thus it is added to the set of security controls to be returned (Line 10). Otherwise, additional security controls must be enforced. Thus, function IdSecurityControls is recursively called over the new set of states S' and violating states S'_v (Line 12). When the set of security controls is returned, each element sc'' of the set of computed constraints sc' is analyzed (Line 13) and enriched with the capability c (Line 14). Note that this procedure can also terminate before all transitions entering in the dark states (S_v) are considered, since some of the violating states targeted by the remaining transitions might have already been removed due to their presence as successors to transitions already considered. The transitions used for pruning the LTS identify a configuration of security controls: the actions indicated by each transition are those for which authorisation should be revoked from the corresponding agent.

However, recall that by removing the action that leads the system to a violating state, the successors of the corresponding homologous transitions are also removed. In other words, not all the transitions entering violating states are necessary for a configuration of security controls. Furthermore, the ordering in which transitions are considered can lead to the identification of different sets of security controls. For this reason, Algorithm 2 returns alternative sets of security controls, for each ordering in which the transitions entering in the violating states (S_v) can be considered.

- In the first scenario (Figure 3a), to satisfy security requirement *SR1*, security controls will forbid $\langle Eve, in, O6 \rangle$ or $\langle Bob, in, O6 \rangle$, depending on whether a transition to state 4 or 6 is considered first, since both of these transitions lead to violating states.
- In the second scenario (Figure 3b), $\langle Mallory, in, O2 \rangle$, will also be included in sets of security controls, to comply with security requirement *SR2*.
- In the third scenario (Figure 3c), following the identification of violating states, $\langle Trudy, in, LT1 \rangle$ and $\langle Trudy, in, LT2 \rangle$ will additionally be forbidden, to comply with security requirement *SR3*.

B. Security Controls Selection

This section illustrates possible criteria for selecting one of the alternative configurations of security controls identified by Algorithm 2. Even though different criteria can be employed, such as minimisation of the number of security controls, in this paper we propose a *requirements-driven* criterion that aims to exclude those configurations of security controls forbidding the satisfaction of non-security requirements (i.e. functional requirements). Since we assume that all the functional requirements have a fixed structure ($EF\ a_g\ in\ l$), which requires the existence of a path that allows an agent a_g to reach location l, we can exploit this structure to detect the set of security controls that do not contrast with these requirements. In other words, the main idea is to compute the set of capabilities that the agent a_g must perform to access l and to filter the security controls that do not remove these capabilities.

Considering our case study and the system functional requirements (7a-7c) we notice that some configurations of security controls violate stated functional requirements. For example, the configurations that revoke from Bob the authorisation to access area O6 violate functional requirement *FR2*, which states that there should always exist a path that allows Bob to reach office O6. To guarantee the satisfaction of this requirement candidate configurations of security controls should never revoke Bob the authorisation to traverse the areas necessary to reach O6 from his location. Note that the security controls selection can be performed together with the identification of security controls. In this case, Algorithm 2 can terminate as soon a configuration of security controls that satisfies all system requirements is identified.

Since it may not always be possible to satisfy all the functional requirements at the same time, a selection criterion can aim to identify a configuration of security controls that satisfies the requirements having highest priority. Another alternative is to aim for less disruption to a specific subset of agents (e.g., those having highest importance), as a kind of quality-of-service principle. For example, to satisfy security requirement *SR1*, potential security controls can revoke from Bob or Eve the authorisation to access the safe room. This happens because both are placed in A2, and can potentially be co-located in O6. If we assume that Bob has a higher priority than Eve, access to O6 will be only revoked to Eve even if she requires to enter O6. Another selection criterion can aim to minimise the number of security controls that are placed in a candidate configuration. This is motivated by the fact that the application of security controls could be expensive (e.g., in terms of energy consumption).

VII. Evaluation

This section evaluates our proposed approach along two dimensions: (I) applicability of the approach and effectiveness of the adaptation procedure, and (II) classes of security requirements that can be handled in the adaptation process.

To evaluate the applicability and efficiency of the approach, we realised it by developing a prototype application. Starting from an Ambient Calculus formula that represents the current topology of the system and its requirements specified using CTL formulae, the application computes the security controls to be employed to protect the system. The implementation includes the set-theoretic procedure described by Mardare et al. [16], [15] and briefly introduced in Section V to generate the LTS, the algorithm to detect and select the security controls to be employed, and uses a third party prototype of the CTL model checking algorithm to identify violations of security requirements. The prototype is realised as a pure Python stand alone application using[4] the MrWaffles CTL checker and NetworkX as a graph library backend.

Preliminary results demonstrate the applicability of the approach and encourage further investigation. For example, for our case study the process took under two seconds on a test machine Intel i5 (2.5GHz) having 4GB RAM. The set of the security controls that is returned by the procedure forbids the following actions: "Trudy *in* LT1", "Trudy *in* LT2", "Mallory *in* O2", "Eve *in* O6". Note that the reference implementation is not optimised with respect to computation time or space; these can be reduced rather significantly, for example by using state of the art model checking tools (e.g., NuSMV) as well as high performance data structures capable to handle large state spaces efficiently. Furthermore, this time can also be reduced by acting on the heuristics used to generate the state space or to select the security controls that can be chosen with respect to the specific case study.

The expressivenes of the proposed approach in terms of security policies that it can enforce was also evaluated. Cuppens and Cuppens [10] classify security policies in four categories: permission, prohibition, obligation and dispensation. In this paper security controls only support permission and prohibition as they grant or revoke from agents the authorisation to perform a specific action. However, our approach is also amenable to support obligation and dispensation. In particular, obligation can be applied by forcing agents to perform actions that lead to a topological configuration that is less close - in terms of number of transitions - to a state where security requirements are violated. Dispensation is conceived as a permission to neglect a security requirement. It could be applied by tolerating some violations of some security requirements. This can be supported by not including during the selection of security controls the states in which a violation of security controls can be tolerated.

VIII. Related Work

The role of topology [23] has been investigated in several domains that are not directly related to software engineering. A representation of topological network connections has also been adopted to manage large scale distributed systems. For example, Tapestry and Pastry construct a topology-aware overlay by choosing nearby nodes for inclusion in their routing tables [8]. Topology has also been extensively taken into account in the wireless sensor networks community, where it is used not only for sensor area placement [14], but also in the context of security. For example, adaptation in sensor network clustering in response to spam attacks has been proposed [12].

However, as far as we are aware, from a software engineering perspective, explicit focus on topology to support adaptive security has not been considered. In particular, Salehie et al. [21] proposed a requirements-driven approach for dynamically re-estimating the risk of harm depending on assets and context changes. Predetermined security controls are also adjusted at runtime depending on the varying risk of harm. Architecture-based self-protection (ABSP) [24] aims to detect and mitigate security threats based on an architectural representation of the software that is kept in sync with the running system. The architectural model provides information related to the impact of a security breach on the system and allows engineering security controls by applying specific architectural design patterns. However, this work does not take into account topological changes as a trigger for adaptation and is based on the assumption that security controls are predetermined. Our approach, instead, monitors changes in topology and reasons about their impact on the satisfaction of security requirements. This allows us to discover new security threats and deploy security controls that have not been previously planned for.

Existing work on dynamic access control has considered contextual information to adapt security policies. Samuel et al. [22] use contextual parameters, such as time and location to identify emergency situations for which users' authorisation might require to be relaxed or made stricter. Similarly, the Organization Base Access Control (OrBAC) [10] allows associating security rules with conditions expressed on time, user's location and previous behavior. These conditions provide the flexibility to enable/disable security rules dynamically. Unlike existing access control approaches, our work avoids specifying security rules in advance. It looks ahead at potential users' actions that can be performed from the current topology and identifies undesired states where security requirements can be

[4] mrwaffles.gforge.inria.fr, networkx.lanl.gov

violated. Adequate security rules are identified dynamically by revoking from relevant agents the right to perform actions that can lead the system to undesired states.

The Ambient Calculus has been extended to represent security relevant properties. For example, ambient types [7] can be used to identify confidential ambients that cannot be opened. Boxed ambients [5] delimit the perimeter within which communication can take place, while boundary ambients [2] prevent information leakage. However, the decision to assign a specific type to an ambient in order to satisfy specific security properties is pre-determined and cannot be modified at runtime. Conversely, our approach is able to detect whether to allow or forbid the execution of agents' capabilities depending on potential threats brought by the current topology.

Several analysis techniques have been proposed in literature to verify if a system modelled as a set of Ambient Calculus formulae satisfies some security requirements. For example, Nielson et al. [11], [19] propose a technique to check whether an Ambient Calculus specification allows only some processes to be contained into other ones. This technique has also been employed for checking if the behaviors of firewalls are compliant with their security policies [18]. Braghin et al. [2], [3] define a verification technique able to identify violations of confidentiality determined when sensitive data can be moved outside a boundary ambient. Other work [4] verifies whether security policies expressed according to Bell-LaPadula model are satisfied in a program that leverages a specific network configuration. However, all the proposed model checking techniques have only been employed to verify security policies but they have not been adopted to assess security risks or to suggest possible security controls that can be applied for specific topologies of the operational environment.

IX. Conclusions

This paper has proposed an approach aimed to engineer topology aware adaptive security systems. A live model of the topology of the operational environment is used at runtime to look ahead at changes which represent future system states. Potential violations of security requirements are identified in future topological configurations reachable from the current one. A set of security controls is proactively applied to prevent the system from reaching those states where security requirements can be violated. Our approach leverages existing formalisms such as the Ambient Calculus to represent the topology and CTL to represent system requirements. Preliminary results demonstrate the viability of the approach in identifying security requirements violations and appropriate security controls for a realistic physical access control scenario. We aim to improve the expressiveness of our approach in order to manage obligation and dispensation security policies, and ameliorate the performance by building on state of the art model checkers. We believe that the approach presented contributes to opening new dimensions in reasoning on systems where topology is a first class entity. Whereas so far the focus has been on physical systems, we aim to apply this initial approach to digital topologies to prevent threats that can arise in digital and cyber-physical systems.

References

[1] C. Bolchini, C. A. Curino, E. Quintarelli, F. A. Schreiber, and L. Tanca. A Data-oriented Survey of Context Models. *SIGMOD Record*, 36(4):19–26, 2007.

[2] C. Braghin, A. Cortesi, and R. Focardi. Security Boundaries in Mobile Ambients. *Computer Languages, Systems & Structures*, 28(1):101–127, 2002.

[3] C. Braghin, A. Cortesi, and R. Focardi. Information Flow Security in Boundary Ambients. *Information and Computation*, 206(2):460–489, 2008.

[4] C. Braghin, N. Sharygina, and K. Barone-Adesi. A Model Checking-Based Approach for Security Policy Verification of Mobile Systems. *Formal Aspects of Computing*, 23(5):627–648, 2011.

[5] M. Bugliesi, G. Castagna, and S. Crafa. Boxed Ambients. In *Proc. of the 4th Int. Symp. of Theoretical Aspects of Computer Software*, pages 38–63, 2001.

[6] L. Cardelli and A. D. Gordon. Mobile ambients. In *In Proceedings of POPL'98*. ACM Press, 1998.

[7] L. Cardelli and A. D. Gordon. Types for Mobile Ambients. In *Proc. of the 26th Symp. on Principles of Programming Languages*, pages 79–92, 1999.

[8] M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron. Topology-Aware Routing in Structured Peer-to-Peer Overlay Networks. In *Future Directions in Distributed Computing*, pages 103–107. Springer, 2003.

[9] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model checking*. MIT press, 1999.

[10] F. Cuppens and N. Cuppens-Boulahia. Modeling contextual security policies. *Int. Journal of Information Security*, 7(4):285–305, 2008.

[11] R. R. Hansen, J. G. Jensen, F. Nielson, and H. R. Nielson. Abstract Interpretation of Mobile Ambients. In *Static Analysis*, pages 134–148. Springer, 1999.

[12] C.-T. Hsueh, Y.-W. Li, C.-Y. Wen, and Y.-C. Ouyang. Secure adaptive topology control for wireless ad-hoc sensor networks. *Sensors*, 10(2):1251–1278, 2010.

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

[14] M. Kwon and S. Fahmy. Topology-Aware Overlay Networks for Group Communication. In *Proc. of the 12th Int. Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 127–136, 2002.

[15] R. Mardare and C. Priami. Computing the Accessibility Relation for the Ambient Calculus. 2003.

[16] R. Mardare, C. Priami, P. Quaglia, and O. Vagin. Model checking biological systems described using ambient calculus. In *Computational Methods in Systems Biology*, pages 85–103. Springer, 2005.

[17] R. Milner. The polyadic pi-calculus: a tutorial. Technical report, Logic and Algebra of Specification, 1991.

[18] F. Nielson, H. R. Nielson, R. R. Hansen, and J. G. Jensen. Validating firewalls in mobile ambients. In *CONCUR99 Concurrency Theory*, pages 463–477. Springer, 1999.

[19] H. R. Nielson and F. Nielson. Shape analysis for mobile ambients. In *Proceedings of the 27th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 142–154. ACM, 2000.

[20] L. Pasquale, C. Ghezzi, C. Menghi, C. Tsigkanos, and B. Nuseibeh. Topology Aware Adaptive Security. In *Proc. of the 9th Int. Symp. on Software Eng. for Adaptive and Self-Managing Systems*, 2014.

[21] M. Salehie, L. Pasquale, I. Omoronyia, R. Ali, and B. Nuseibeh. Requirements-Driven Adaptive Security: Protecting Variable Assets at Runtime. In *Proc. of the 20th Int. Requirements Engineering Conf.*, pages 111–120, 2012.

[22] A. Samuel, A. Ghafoor, and E. Bertino. Context-Aware Adaptation of Access-Control Policies. *IEEE Internet Computing*, 12(1):51–54, 2008.

[23] J. Wang and G. M. Provan. A Comparative Analysis of Specific Spatial Network Topological Models. In *Proc. of the 1st Int. Conf. on Complex Sciences*, pages 1514–1525, 2009.

[24] E. Yuan, S. Malek, B. R. Schmerl, D. Garlan, and J. Gennari. Architecture-Based Self-Protecting Software Systems. In *Proc. of the 9th Int. Conf. on Quality of Software Architectures*, pages 33–42, 2013.