Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2013) 21: 144 – 173

doi:10.3906/elk-1105-54

Spatiotemporal model checking of location and mobility related security policy specifications

Devrim ÜNAL[1,*], **Mehmet Ufuk ÇAĞLAYAN**
[1]TÜBİTAK BİLGEM, Gebze, Kocaeli, Turkey
[2]Department of Computer Engineering, Boğaziçi University, Bebek, İstanbul, Turkey

Received: 29.05.2011 • **Accepted:** 15.08.2011 • **Published Online:** 27.12.2012 • **Printed:** 28.01.2013

Abstract: For the formal verification of security in mobile networks, a requirement is that security policies associated with mobility and location constraints are formally specified and verified. For the formal specification and verification of security policies, formal methods ensure that a given network configuration that includes certain network elements satisfies a given security policy. A process calculus based approach is presented, where ambient calculus is used for formal specification of security policies and ambient logic is used for formal representation of mobility and location constraints. A spatiotemporal model checking algorithm is presented for the model checking of formal specifications in ambient calculus with respect to formulas in ambient logic. The presented algorithm allows spatiotemporal model checking of security policy rules and consists of spatial and temporal model checking algorithms. The spatial model checking algorithm is implemented in the Java language and the temporal model checking algorithm is implemented using the NuSMV model checker.

Key words: Security policy, model checking, ambient calculus, spatiotemporal

1. Introduction

Providing security services in multiple administrative domains is an increasing need, as well as a challenge. In mobile networks with multiple domains, users are allowed to roam among domains. Roaming users may connect to multiple domains, and they may use resources in these domains. Management of interdomain security policies should be supported for security management in multiple domain networks. Authorization mechanisms are used for determining the access rights for a user, with respect to a security policy. The actions of users should be verified with respect to interdomain policies while they are accessing resources on multiple domains. We can utilize formal verification in order to ensure that security mechanisms are not bypassed and the security policy is not violated by the roaming users, which would result in an abuse of the internal trust relationships.

In mobile networks with multiple domains, security policies associated with mobility and location constraints should be formally specified and verified. The existence of mobile users in this environment presents challenges for formal specification and formal verification. Formal methods provide means for ensuring that the design of a system is consistent with its specification. Formal methods are used within the context of security policy specification and verification, for the purpose of ensuring that a given configuration of a network satisfies the security policy.

We are concerned with formal specification of security policy in an environment where users roam within

*Correspondence: devrim.unal@tubitak.gov.tr

a network that contains different administrative domains. A security policy determines the allowed actions for subjects to conduct on objects and defines the conditions for the access. An administrative domain defines a set of subjects and objects. Actions related to mobility determine the functions that may be conducted by a subject in an administrative domain with respect to locations and mobility within the network. Following these definitions, we formalize our problem definition as follows: "Given a model of a network that includes mobile users roaming different administrative domains, to determine formally whether the actions related to mobility are compliant with the security policy". Our purpose is to verify compliance formally, in terms of a formal specification of security policy with respect to a formal representation of location constraints together with mobility constraints.

In this study, the formal security policy model is based on ambient calculus and the formal representation of mobility and location constraints is based on ambient logic. Multidomain mobile networks have multiple locations and mobile users. Therefore, the mobility and location of users are considered as a part of the security policy in our proposed method. Our method is based on process calculus and modal logic. Ambient calculus is a process calculus aimed at mobility and location of processes. Ambient logic is a type of modal logic, which is used for reasoning about spatiotemporal properties related to ambient calculus processes. The satisfaction relation is evaluated through spatiotemporal model checking, for verification of mobility and location constraints with respect to security policies in a given mobile network configuration.

In this paper, a spatiotemporal model checking algorithm is presented for the model checking of ambient calculus specifications with respect to ambient logic formulas. The presented algorithm allows spatiotemporal model checking of security policy rules. The spatiotemporal model checking algorithm includes 2 phases, namely spatial model checking and temporal model checking.

The model checking algorithm that we propose introduces heuristics for spatial model checking in order to reduce the time and space complexity. We presented a preliminary version of this algorithm in [1]. In this paper, we present the complete algorithm and provide an extended analysis of performance and complexity based on a case study. The spatiotemporal model checking algorithm has been implemented as a tool called the ambient calculus model checker. The spatial model checking phase is implemented in the Java language. For the temporal model checking phase, the NuSMV model checking tool has been used.

We present related work and background information on ambient calculus and ambient logic in Section 2. We present a formal model for security policies with mobility and location aspects in Section 3. We propose a novel spatiotemporal model checking algorithm for ambient calculus in Section 4. We present a complexity and performance analysis of the proposed algorithm based on a case study in Section 5. We conclude with Section 6.

2. Background and related work

2.1. Formal methods and languages for the specification and verification of security policies

Logic based models for security policies have been used as a general framework for formal specification of security policies. SECPAL [2] is a logic based policy language suitable for security of grid computing. A logic programming based method named Flexible Authorization Framework (FAF) [3, 4] provides a framework for authorization policies that supports formal specification, derivation, and conflict resolution. Another logic based model [5] supports explicit denial rules, policy hierarchies, derivation of new policies, and resolution of conflicts. Ponder [6] is a policy language for Java, which is suitable for network services security management. Woo and Lam [7] proposed a formal language where authorizations are specified in a declarative manner, based

on the concept of policy bases. Deontic logic was used in [8] for modeling organizational security policies. A set-and-function formalism based security policy language was presented in [9]. A formal model of security in mobile ad hoc networks based on cryptography was studied in [10].

Model checking and theorem proving are 2 popular formal methods used for verification of security policies. The first-order logic based Acpeg tool [11] supports access control policy evaluation and generation. In our previous study [12], we presented a theorem proving based approach where the calculus of inductive constructions is used as a formal basis for specification of a formal security policy model that supports automated conflict detection. RBAC authorization constraints are formalized and verified based on first-order linear temporal logic using the Isabelle theorem prover in [13]. Nontemporal and history based constraints were formalized in [14] based on first-order linear temporal logic (LTL) and the Object Constraint Language (OCL), and some RBAC policies were verified using a theorem prover.

Ambient calculus [15] is a process calculus with mobility and location primitives. Regarding mobile systems, ambient calculus is also used for formal representation and reasoning of security issues. Spatial modeling of security policies in the application layer, based on a version of ambient calculus, was investigated in the PhD thesis of Scott [16]. $\mathrm{BACI_R}$ is a version of boxed ambient calculus applied to RBAC mechanisms for the purpose of defining access control policies for ambients in [17].

In our study, the ambient calculus process calculus specification language [18] and the ambient logic [19] modal logic are also used. Process calculus specifications with ambient calculus are used for modeling the mobile network and the associated security policies. Ambient logic is used for formal specification of mobility and location constraints within security policies. Predicate logic is used for modeling the relational aspects. In our model, locations denote logical boundaries of domains and hosts. In contrast to our approach, $\mathrm{BACI_R}$ deals with policies embedded within ambient calculus formulas, and Scott focused on application level policies in which locations denote physical location boundaries.

Charatonik et al. [20, 21] proposed an exhaustive search based model checking algorithm for ambient calculus. Mardare et al. [22] proposed an algorithm based on time and space decomposition. Our proposed algorithm includes 2 main contributions that complement and enhance existing model checking algorithms for ambient calculus. First, we propose a novel data structure called capability trees. This data structure is used in the state space generation operation in order to reduce the complexity. The matching operation of ambient logic spatiotemporal formulas to the state space is also improved. Second, we offer heuristics for checking spatial modalities in order to reduce the search space.

2.2. Ambient calculus and ambient logic

Ambient calculus was proposed by Cardelli and Gordon [15]. It is a process calculus for concurrent systems that includes mobility and location primitives. We use ambient calculus to model the mobility and location aspects of mobile networks and their associated security policies. We use a version of the ambient calculus language, which is decidable for model checking. The fragment that we use in this study is listed in Table 1.

Like all process algebras, ambient calculus relies on the notion of process. Processes are the counterparts of real world system elements. Properties of ambient calculus processes can be analyzed as spatial properties and temporal properties. The notion of ambient is the basic spatial element. Ambients are bounded places identified by a name where processes reside inside or outside. Ambients can be nested in other ambients. This provides hierarchical organization of locations.

All process algebras have constructs that resemble the changes of the real world systems over time,

Table 1. Mobility and location primitives within ambient calculus.

P, Q	::=	process	M	::=	capability
	0	inactivity		x	variable
	$P\|Q$	composition		n	name
	$M[P]$	ambient		$in \quad M$	can enter M
	$M.P$	capability		$out \quad M$	can exit M
	$(x).P$	input		$open \quad M$	can open M
	$\langle M \rangle$	output		ϵ	null
				$M.M$	path

called temporal constructs. Most of the calculi provide temporal constructs as communication primitives. Because the main aspect of ambient calculus is modeling mobility, it provides temporal constructs in addition to communication primitives. There are 3 main temporal constructs in ambient calculus: *in n*, *out n*, and *open n*. Capabilities can represent sequential execution as well as parallelism. Their effect on ambients is regulated by the name references they contain. Capabilities are ordered as sequences called paths. They must be attached to an ambient or inactivity (*0*). While the ability of change of ambient hierarchy represents mobility, it can be used to represent any kind of computation. There are also communication primitives for exchanging messages between processes within the same ambient.

Names are used for identifying ambients as well as for accessing ambients; capabilities can affect or use ambients whose names are known by them. Two distinct ambients can have the same name, but in the fragment of ambient calculus in this paper, names are restricted to be unique. They become an identifier in the scope of this paper. This restriction reduces the implementation complexity by eliminating renaming and bookkeeping tasks for ambients, which are not core model checking tasks.

The inactive process, 0, specifies the empty process, which does nothing. It is not reducible. Parallel execution of the processes is represented by the parallel composition operator. It is a commutative and associative operator. In ambient calculus, communication constructs are asynchronous. Communication is local to an ambient. Ambient calculus does not support channel names for communication. Both names and capabilities may be communicated in the full fragment of the calculus. However, in this paper communication of capabilities is excluded.

Variables are place holders for names when an input operation is included in a capability path. When an output operation provides a name to an input operation, every instance of variables in the scope of input capability is replaced with the incoming name. The semantics are defined based on the structural congruence relation. Structural congruence defines processes by elementary spatial rearrangements. The dynamic properties of ambient calculus originate from capabilities and communication primitives. This set of constructs is called temporal constructs. The semantics of these constructs are identified by reduction relations.

As mentioned before, modal logics are used for expressing properties of models that cannot be expressed by the constructs of calculi. Ambient logic expresses spatiotemporal constructs of ambient calculus processes. The main differences of ambient logic from latter logics are more expressive space modalities and simpler temporal connectives [23, 24].

For the purposes of this study, a fragment of ambient logic has been used and the syntax of this fragment

Table 2. Syntax of ambient logic.

η		a name n		
$\mathscr{A},\mathscr{B},\mathscr{C}$ ::=	T	true	$\mathscr{A}\|\mathscr{B}$	composition
	$\neg\mathscr{A}$	negation	$n[\mathscr{A}]$	location
	$\mathscr{A}\vee\mathscr{B}$	disjunction	$\Diamond\mathscr{A}$	sometime modality
	0	void	$\diamond\mathscr{A}$	somewhere modality

is given in Table 2. Spatiotemporal modalities as well as propositional logic constructs are included in the ambient logic. The semantics are defined by satisfaction relations. The satisfaction relation, denoted by $\models$, is defined with respect to structural congruence of processes. $P \models \mathscr{A}$ denotes that P satisfies $\mathscr{A}$, where P is a process and $\mathscr{A}$ is a formula. Π is the set of processes, Φ is the set of formulas, ϑ is the set of variables, and Λ is the set of names.

Ambient logic provides 2 main spatiotemporal constructs. The *Somewhere* connective, $\diamond$, is used for specifying nesting properties of processes. The formula $\diamond\mathscr{A}$ is satisfied by processes that satisfy A in some inner location. The *Sometime* connective, $\Diamond$, is used for specifying temporal behavior of the processes on the basis provided by reduction relations ($\rightarrow$).

3. The formal model of security policies with mobility and location constraints

The formal security policy model consists of 2 formal models related to mobility and locations. First, a formal model of *mobility and location constraint* is required. Second, a formal model of actions and the location configuration of the system are required. Because of the dynamic nature of mobility, in mobile networks the location (spatial) configuration of the system changes with actions. This necessitates the use of time in the policy model. Ambient calculus is used for representation of actions and the location configuration. Time and location aspects of security policy rules are modeled as mobility and location constraints. We utilize ambient logic for specification of mobility and location constraints.

3.1. The access control model

The access control model is defined by a formal specification of an extension of the role based access control (RBAC) [25] model. The predicate calculus and first order set theory formalisms are the basis of the formal specification. In order to introduce mobility and location constraints into security policies, we propose an extension to the RBAC model, which consists of the set of *domains*, the set of *hosts*, the set of *object types,* and a formal authorization term structure including *conditions* and *location constraints.*

- Constants in the model:

  d: Number of domains, n: number of hosts, m: number of users, o: number of roles, t: number of objects, and v: number of object types.

- Sets in the model:

  - $\Delta = \{D_1, D_2, ...D_d\}$: Domains , $H = \{H_1, H_2, ...H_n\}$: Hosts.
  - $U = \{U_1, U_2, ...U_m\}$: Users , $R = \{R_1, R_2, ...R_o\}$: Roles.

- $O = \{O_1, O_2, ..., O_t\}$: Objects , $OT = \{OT_1, OT_2, ...OT_v\}$: Object types.
- $AS = U \cup R$: Authorization subjects, the set of active entities that are the subjects of an action.
- $AO = O \cup OT \cup H \cup D$: Authorization objects, the set of passive entities that are the objects of an action.
- A : Set of actions. A ={*Enroll, Login, Logout, Execute, Read, Write, Send, Receive, Delete, Create*} .
- Signs: $S = \{+, -\}$. A signed action $sa \in S \times A$ denotes permission (+) or denial (-) for an action.

- Relations in the model:

  $HOD : H \times \Delta$: Mapping for hosts to domains. $HOD(H_i, D_a)$ denotes enrollment of host H_i to domain D_a.

  $UOD : U \times \Delta$: Mapping for users to domains. $UOD(U_j, D_a)$ denotes enrollment of U_j to domain D_a.

  $OOT : O \rightarrow OT$:A function returning the object type. $OOT(O_k)$ is the type of the object O_k.

  $UA : U \times R$: User-role assignment relation.

  $PA : R \times AO \times A$: Role permission association relation.

- Conditions in the model are first order predicate logic sentences, and the following predicates are predefined:

  1. EDR $(\mathcal{D}_i, u_j)$, where $\mathcal{D}_i \in \Delta, u_j \in U$ specifies whether a user u_j has been enrolled (registered) to domain $\mathcal{D}_i$.
  2. EDH $(\mathcal{D}_i, h_k)$, where $\mathcal{D}_i \in \Delta, h_k \in H$ specifies whether a host h_k has been enrolled (registered) to domain $\mathcal{D}_i$.
  3. ADU $(\mathcal{D}_i, u_j)$, where $\mathcal{D}_i \in \Delta, u_j \in U$ specifies whether user u_j has been logged into domain $\mathcal{D}_i$.
  4. RSG (u_j, r_l), where $u_j \in U, r_l \in R$ specifies whether user u_j has rights to assume role r_l.
  5. RAS (u_j, r_l), where $u_j \in U, r_l \in R$ specifies whether user u_j has actively assumed role r_l.
  6. REN $(r_l, \mathcal{S}_m)$, where $\mathcal{S}_m \subset AO, r_l \in R$ specifies whether role r_l is enabled for a subset of authorization objects $\mathcal{S}_m$.
  7. DR (r_l, r_k), where $r_l, r_k \in R$ specifies whether a given role r_l is descendant (or specialization) of another given role r_k.
  8. OIT (o_n, t_m) specifies whether an object $o_n \in O$ identified by its object name is of a given object type $t_m \in OT$.
  9. ADM $(u_j, \mathcal{D}_i)$, where $u_j \in U, \mathcal{D}_i \in \Delta$ specifies whether user u_j has administrative rights over domain $\mathcal{D}_i$.

- Spatial formula is an ambient logic formula including domains, authorization subjects, and authorization objects.

$at = (as,\ ao,\ sa,\ fo,\ co)$ is an *authorization term*, where $as \in$ AS, $ao \in$ AO, $sa \in S \times A$, fo (formula) is a formal specification in ambient logic, and co (condition) is a formal specification in predicate logic. A set of authorization terms defines a *security policy*.

3.2. Mobility and location model

The mobility and location formal model provides a spatiotemporal model for security policies. Mobility and location constraints are represented by this formal model. There are 2 main uses of the mobility and location model. First, a formal specification in ambient calculus represents the current location state of the system. Second, a location formula specified as an ambient logic formula represents the mobility and location constraints. Spatiotemporal model checking is used to evaluate the satisfaction of mobility and location constraints with respect to the current state of the system. Spatiotemporal model checking is based on evaluation of the satisfaction relation, which has been defined in Section 2.2.

LCONF, the location configuration of the system, is a formal representation of the current system state. The initial state $LCONF_0$ is generated from the network configuration and the security policies. Whenever a user requests to execute an action, that action is translated into a formal specification. The action may be executed only if the action is allowed by the security policy. The location configuration is regenerated according to the new system state. An example location configuration is as follows:

$$LCONF = D_1[U_1[in\ H_1.0|out\ H_1.0|in\ F_1.0|out\ F_1.0]|H_1[F_1[Data_1[in\ U_1.0|out\ U_1.0]]]].$$

This example includes a single domain, D_1. The user U_1 has a read permission to the file F_1. U_1 logs into the host H_1 in order to read $Data_1$ from the file F_1. The formalization of actions is handled by translating them to ambient calculus capabilities. The parallel (|) operator is used to combine multiple actions. Any of the combined actions may be exercised in an independent fashion. Spatiotemporal model checking is applied to determine whether a location constraint fo is satisfied in the current location configuration $LCONF$, i.e. $LCONF \models fo$.

The authorization term is defined as a 5-tuple $at = (as, ao, sa, co, fo)$. An authorization term formally represents a security policy rule. Here, as is an authorization subject, ao is an authorization object, sa is a signed action, co is a generic constraint, and fo is a mobility and location formula. A generic constraint is specified as a predicate calculus formula, whereas the mobility and location formula is specified as an ambient logic formula.

We now present an example of formal specification of policy rules with mobility and interdomain access constraints. In this example, $\{UniA, UniB\} \subset \Gamma, u \in U, Student \in R, p \in O, Portable \in T$. In this specification, *UniA* represents the domain belonging to University A, and *UniB* represents the domain belonging to University B. The corresponding formal specification for the example policy rule, "The students of University B are permitted to connect portable computers to the network of University A", is as follows: $at = (as = Student, ao = p, sa = +connect, fo = "World[UniB[T]|p[u[T]|T]|UniA[T]|T]", co = "(EDR(u, UniB) \wedge RAS(u, Student) \wedge OIT(p, Portable)")$. In order to derive a formal specification from the written security policy rule, one may first provide a more detailed interpretation of the written policy rule, such as the following. There exists a user who has enrolled in *UniB* domain and who has assumed the *Student* role, and is logged into a computer named p, which has the object type *Portable*. The user has not connected the *Portable* computer to either the *UniA* or *UniB* domain. The user is currently in a location that gives the possibility to connect to either the *UniA* or *UniB* domain. In this case the user may connect the *Portable* computer.

Table 3. Ambient calculus encoding for actions in security policy rules.

Action	Ambient calculus specification	Action	Ambient calculus specification
$\mathbf{Enroll}_{z,d}$	$new\ z.z[in\ d] \mid d[]$	$\mathbf{Login}_{u,l}$	$u[in\ l] \mid l[]$
$\mathbf{Logout}_{u,l}$	$l[u[out\ l]]$	$\mathbf{Send}_{u_1,m}$	$d[h_1[u_1[m[M\|out\ u_1.out\ h_1.in\ h_2.$ $in\ u_2]]]\|[h_2[u_2[]]]]$
$\mathbf{Receive}_{u,m}$	$u[open\ m.(m)\|m[M]]$	$\mathbf{Execute}_{u,prog}$	$d[h[u[open\ prog] \mid prog\ [in\ u.P]]]$
$\mathbf{Read}_{u,f}$	$h[f[data[in\ u]] \mid u[in\ f]]$	$\mathbf{Write}_{u,f}$	$h[f[T] \mid u[in\ f.data[out\ u.0]]]$
$\mathbf{Delete}_{u,file}$	$h[file[in\ u] \mid u[open\ file.0]]]$	$\mathbf{Create}_{u,file}$	$h[u[new\ file.file[out\ u]]]$

3.3. Formal specification of rules in the security policy

Each domain, host, user, and object is modeled as an ambient. Actions are formalized as ambient calculus capabilities. A set of ambient calculus process specifications represents the current state of the mobile network. The process specifications, which include capabilities, objects, and ambients, are checked with respect to the security policy. An ambient may *input* or *output* resources. There are various types of ambients such as world, domain, host, and user. We now present some examples for formal specification of object, host, and user mobility.

The following notation conventions are used: $n[]$ denotes $n[0]$, $\rightarrow$ denotes process reduction, and $\rightarrow^*$ denotes a series of process reductions.

- $File_1$ is moved to $Portable_1$:
  $World[Domain_A[Server_1[folder[out\ folder.out\ Server_1.in\ Portable_1.in\ folder.File_1[]]]|$
  $Portable_1[folder[]]]] \rightarrow^* World[Domain_A[Server_1[folder[]|Portable_1[folder[File_1[]]]]]$

- A message M is sent from $User_1$ to $User_3$:
  $World[Domain_A[Server_1[User_1[message[M|out\ User_1.out\ Server_1.out\ Domain_A.in\ Domain_B.$
  $in\ Client_2.in\ User_3.0]]]]|Domain_B[Client_2[User_3[open\ message.(m).0]]]] \rightarrow^* World[Domain_A$
  $[Server_1[User_1[]]]|Domain_B[Client_2[User_3[M]]]]$

We now present a formal mapping of security policy actions into ambient calculus specifications. There are template mappings for each kind of action. For a specific security policy and network configuration, a formal specification in ambient calculus is generated as an input to the ambient calculus model checker. In order to generate this specification, the names for subjects and objects in the system are derived from the security policy and network configuration. The method for generating ambient calculus specifications will be explained in the next section.

Ambient calculus encoding for actions in security policy rules are specified in Table 3. In this encoding, $\alpha_{as,ao}$ represents an action with authorization subject as and authorization object ao, and $z \in U \cup H, u, u_1, u_2 \in U, d, d_1, d_2 \in \Delta, h, h_1, h_2 \in H, l \in H \cup \Delta, m, prog, file, data \in O,$ and M, P are process specifications.

Ambient logic is used to specify the spatial formula in the authorization term. There is a possibility of modal conflicts when actions that have opposite modalities (i.e. permission and denial) but have the same subject and object are defined within the same policy. These conflicts are detected and resolved by theorem proving, as presented in our previous work. Rules are presented to the model checker after conflicts are resolved. An authorization term defines a security policy rule including mobility and location constraints. The process

of formalizing authorization terms and spatial formulas has been described in this section. Examples of formal security policy rules, which are derived by the formalization process, are presented below.

- All users are permitted to read the files in the folder named *Projects*, provided that they are within a location that contains the *Projects* folder: (as = user, ao = *Projects*, sa = + *read*, fo = $\diamond(as[] \mid ao[])$, co = $user \in U \wedge OIT(ao, Projects)$).
- All users are permitted to send *E-mail* messages among the *UniA* and *UniB* domains: (as = user, ao = *E-mail*, sa = + *send*, fo = *UniA* $[\diamond as[]] \mid$ *UniB* $[\diamond ao[]] \vee$ *UniB* $[\diamond as[]] \mid$ *UniA*$[\diamond ao[]]$, co = $user \in U \wedge OIT(ao, E-mail)$).

The model checker inputs the formal specification of security policy rules, which consists of an ambient calculus specification and ambient logic formulas. The model checker implements the spatiotemporal model checking algorithm. The security policies are specified by the ambient calculus and ambient logic formal specifications. The location configuration of the system is specified using ambient calculus. The mobility and location constraints within security policy rules are specified using ambient logic. We present an example network configuration and a security policy rule, which includes a location configuration and mobility and location constraints. Two domains exist in this example: $Domain_1$ and $Domain_2$. $User_2$ is a mobile user of $Domain_2$. $User_2$ reads data from $File_1$ after logging into host $Host_1$ in domain $Domain_1$. The ambient logic formula states that $Data_1$ and $Data_2$ may not reside together within $Host_2$. This security policy rule indicates that $Domain_2$ data should not be copied to $Domain_1$.

- Ambient calculus specification: $Domain_1[User_1[]|Host_1[File_1[Data_1[in\ User_2.0|out\ User_2.0]]]]$ $|Domain_2[Host_2[User_2[out\ Host_2.0|out\ Domain_2.0|in\ Domain_1.in\ Host_1.0|out\ Host_1.out\ Domain_1.0$ $|in\ Domain_2.in\ Host_2.0|in\ File_1.0|in\ File_2.0|out\ File_1.0|out\ File_2.0]|File_2[Data_2[]]]]$.
- Ambient logic formula: $\Box\{\neg \diamond \{\diamond Host_2[\diamond\{Data_1[T]|Data_2[T]\}]|T\}$.

3.4. Translation of security policies into formal specifications

Formal specifications are automatically extracted from security policies using the method that is presented in this section. The translation process is carried out by formalization of the actions within security policy rules. Multiple domain and interdomain security policies are processed and a single formal specification is generated. In this context, our method includes 2 steps:

1 – The domain and interdomain security policies are translated to formal ambient calculus specifications.

2 – The mobility and location constraints are translated to formal ambient logic specifications.

The security policies and the domain configurations are specified in XML. A domain configuration consists of data sets in addition to relations of a domain, as presented in Section 3.1. In the first step, XML Schema Translation (XSLT) is utilized for translation of security policies and domain configurations to ambient calculus specifications. The resulting ambient calculus specification is called the location configuration (LCONF). The location configuration represents the current state of the mobile network and the associated security policy. The state information consists of location hierarchies and actions within the security policy rules. Multiple XML files representing the network configuration and security policies are used to generate LCONF. The domain and interdomain configurations together with security policies are processed and transformed into an ambient calculus specification using the XSLT processing language. Another XSLT transformation is further applied to

this specification and the location configuration LCONF is generated. This transformation is based on BNF. The overall process is depicted in Figure 1.

In the second step, the location formulas are translated into ambient logic specifications. This translation is achieved by applying XSLT transformations to XML definitions of security policy rules. An XML element is defined for each construct in ambient logic. The BNF notation for ambient logic is taken as a basis for the definition of XML elements within an XML schema.

An XML specification corresponding to the XML schema describing the ambient calculus and ambient logic syntax is translated to a formal specification corresponding to the respective BNF notations. The XSLT *template* structure is used for this purpose. In this template, there is a *for each* construct corresponding to each of the elements in the XML schema. The details of the transformation process, including the XML specifications and XSLT transformations, may be found in our related work [26].

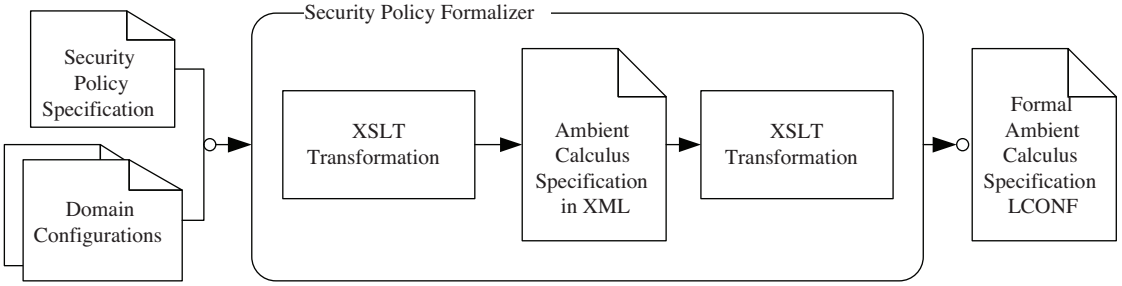

Figure 1. Generating ambient calculus specifications from security policy.

4. Model checking of security policies with ambient calculus

The main idea in model checking is to define an automated mechanism that explores all possible states of a system and tests these states based on a set of desired properties. For a specific system, the set of all possible states must be finite for such an exploration to be performed. There are 3 main processes in model checking, namely modeling, specification, and verification. The abstract system description, which is called a model, represents the system with an acceptable size of state space. The specification process of model checking consists of describing the model and its properties by a formal language. Verification process of model checking can be defined as exploring all states in a model and checking whether a certain set of properties is valid for these states. The process is an exhaustive search, where all reachable states must be visited.

4.1. Model checking for security policies

We apply model checking for verification of security policies. This methodology involves representation of the network state as process calculus specifications. The representation includes the future evolution of the network state in terms of events. The specification is called the location configuration (LCONF). The mobility and location constraints within the rules of the security policy are formalized as modal logic statements, which are expressed as location formulas. Model checking of process calculus specifications with respect to the modal logic statements is used to determine whether the mobility and location constraints are satisfied by the current network state.

4.1.1. State based representation of security policy

Events are the means of changing the process state by use of actions in the process specifications associated by ambients. In Figure 2, ambient c does an *in* and ambient b does an *out* action. The state of the system changes

in reaction to these actions. Each of them is called an event. In our model, an event represents an action in ambient calculus, which is formally mapped to actions in the security policy.

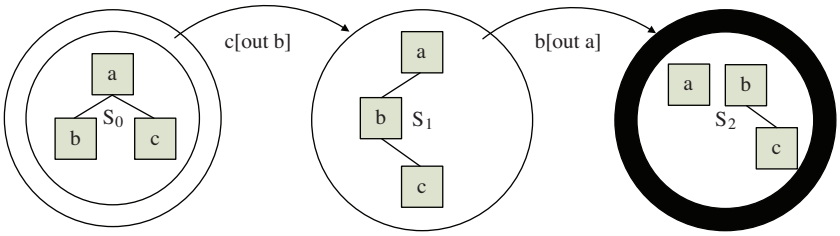

Figure 2. Events as a means to change the state of process specification.

An ambient calculus specification such as $d[in\ c.(R).P]$ may be represented by a *trace*, which can be described as a sequence of events: $in\ c, (R), P$. A *Trace* is a set of statements T where $\{P\}T\{Q\}$ consists of precondition P, which holds in the initial state, and postcondition Q, which holds in the final state.

A set of preconditions and postconditions may represent a security policy rule. For example, for the rule "Project files in $Server_c$ cannot be read by the students", the precondition is $P = \diamond Server_c[data1]$ and the postcondition is $Q = \diamond Student[data1]$. If the model checker finds a sequence of events T such that $\{P\}S\{Q\}$, then the policy rule is applicable. Since the rule indicates denial of access, the access request for a student to read project files in $Server_c$ should be denied in this state.

4.1.2. Finding compliance to security policy by model checking

The formal specification of compliance of a system configuration to security policy is as follows: Find T such that $\{P\}T\{Q\}$. For this purpose, a trace is to be found such that when P holds, trace T is executed and Q holds. The model checker can generate a counterexample T that satisfies $\{P\}T\{Q\}$. This will result in a trace that represents a match of a mobility and location constraint in a security policy rule.

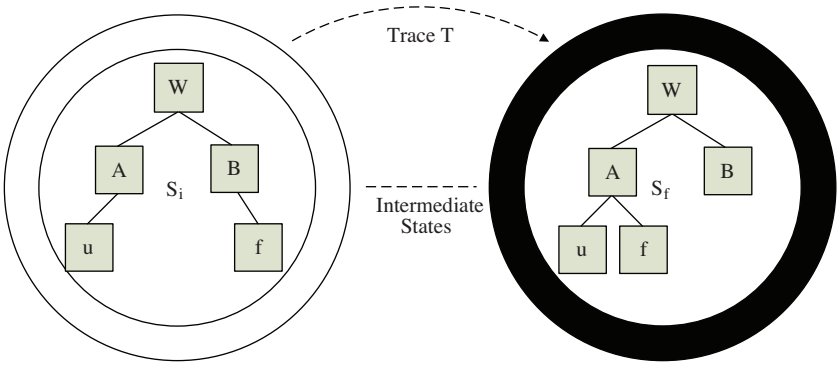

Figure 3. Finding a trace T that leads to a final state from an initial state.

An initial state shows the current execution state of a mobile process specification. After a sequence of events, if a final state is reached where a security policy rule is matched, the security policy rule is applicable in that state. For example, the rule in Figure 3 can be such that "Files in Domain B cannot be copied to Domain A". As a result of the trace T, the file is copied to Domain A. In this case, the security policy rule is applicable and the system should deny the access request. The problem now is to find such a trace T. The model checking algorithm is used for this purpose.

In this paper, an algorithm is proposed for model checking of ambient calculus specifications with respect to ambient logic formulas. The ambient calculus model checking can be decomposed into 2 major search problems. The first problem is to search future evolutions of a given model. A logic formula can include constructs quantifying the rest of the formula over future states. When evaluating the truth of a formula with respect to an ambient calculus specification, analysis of reachable future states is needed. The second problem is to search for spatial congruence of the model and the logic formula. Both ambient calculus specifications and ambient logic formulas have spatial patterns. The model checker must match these patterns in the calculus specifications in order to evaluate the satisfaction relation.

4.2. Formal semantic definitions of ambient logic formulas

Modal logics are used for expressing properties of models that cannot be expressed by the constructs of calculi. Ambient logic is a kind of modal logic, in which spatiotemporal properties of ambient calculus specifications are represented. Here, we provide formal semantic definitions for ambient logic formulas.

Definition 1 *The atomic formula T of ambient logic is satisfied by all processes of ambient calculus.*

$$\forall P \in \Pi . P \models T$$

Definition 2 *Negation of a formula is satisfied by any process that does not satisfy the original formula.*

$$\forall P \in \Pi \mathscr{A} \in \Phi . P \models \neg \mathscr{A} \stackrel{\triangle}{=} \neg P \models \mathscr{A}$$

Definition 3 *A process satisfies the formula $\mathscr{A} \vee \mathscr{B}$ if it satisfies either $\mathscr{A}$ or $\mathscr{B}$.*

$$\forall P \in \Pi, \mathscr{A}, \mathscr{B} \in \Phi . P \models \mathscr{A} \vee \mathscr{B} \stackrel{\triangle}{=} P \models \mathscr{A} \vee P \models \mathscr{B}$$

Definition 4 *The formula 0 is satisfied by only processes structurally congruent to inactivity process.*

$$\forall P \in \Pi . P \models 0 \stackrel{\triangle}{=} P \equiv 0$$

Definition 5 *The formula $n[\mathscr{A}]$ is satisfied by processes within ambients structurally congruent to $n[P']$ for any P' where $\mathscr{A}$ is satisfied by P'.*

$$\forall P \in \Pi, n \in \Lambda, \mathscr{A} \in \Phi . P \models n[\mathscr{A}] \stackrel{\triangle}{=} \exists P' \in \Pi .\ P' \models \mathscr{A} \wedge P \equiv n[P']$$

Definition 6 *The formula $\mathscr{A}|\mathscr{B}$ is satisfied by decomposable processes that may be represented as $P'|P''$, with the condition that P' is a process that satisfies $\mathscr{A}$ and P'' is a process that satisfies $\mathscr{B}$.*

$$\forall P \in \Pi, \mathscr{A}, \mathscr{B} \in \Phi . P \models \mathscr{A}|\mathscr{B} \stackrel{\triangle}{=} \exists P', P'' \in \Pi .\ P \equiv P'|P'' \wedge P' \models \mathscr{A} \wedge P'' \models \mathscr{B}$$

Definition 7 *The nesting relation, denoted by $\downarrow$, is defined over 2 processes as $P \downarrow Q$ and implies that Q is nested 1 level down in any ambient that exists at the top of the topology of P.*

$$P \downarrow P' \quad \text{iff } \exists\, n, P'' . P \equiv n[P']|P''$$

Definition 8 *Relation $\downarrow^*$ is reflexive transitive closure of $\downarrow$. $P \downarrow^* Q$ implies that P contains Q somewhere in its topology.*

$$\downarrow^* \text{ is the reflexive and transitive closure of } \downarrow$$

Definition 9 *The somewhere connective, $\diamond$, is used for specifying nesting properties of processes on the basis provided by Definition 8. The formula $\diamond\mathscr{A}$ is satisfied by processes that satisfy A in some inner location.*

$$\forall P \in \Pi, \mathscr{A} \in \Phi. P \models \diamond\mathscr{A} \stackrel{\triangle}{=} \exists P' \in \Pi.\ P' \models \mathscr{A} \wedge P \downarrow^* P'$$

Definition 10 *The sometime connective, $\Diamond$, is used for specifying temporal behavior of the processes on the basis provided by the reduction relation ($\rightarrow$). The relation $\rightarrow^*$ is reflexive transitive closure of the reduction relation. $\Diamond\mathscr{A}$ is satisfied by processes that can evolve into a future process holding $\mathscr{A}$.*

$$\forall P \in \Pi, \mathscr{A} \in \Phi. P \models \Diamond\mathscr{A} \stackrel{\triangle}{=} \exists P' \in \Pi.\ P' \models \mathscr{A} \wedge P \rightarrow^* P'$$

4.3. The model checking algorithm for ambient calculus

The block diagram of the model checker for ambient calculus is depicted in Figure 4. The model checking process is divided into 2 phases. Temporal model checking is the first phase and spatial model checking is the second phase. The purpose of this decomposition is to benefit from existing methodologies. The evaluation of the satisfaction relation for evaluating the ambient logic temporal connectives *Sometime* ($\Diamond$) and *Everytime* ($\Box$) are carried out in the first phase. In the second phase all the reachable future states are generated and the state transition system (STS) is constructed by spatial model checking. The spatial model checking algorithm evaluates ambient logic formulas in each state. A Kripke structure (Definition 14) is generated by this algorithm.

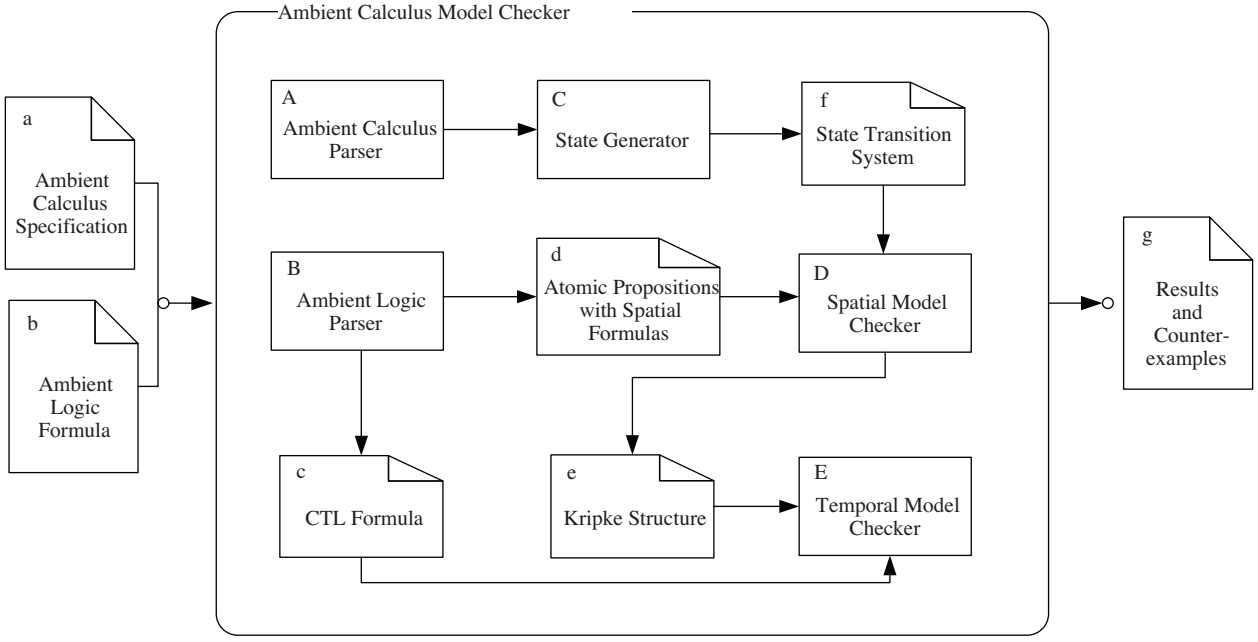

Figure 4. Structure of the model checker for ambient calculus.

The temporal model checking algorithm, which is implemented using the NuSMV [27] model checker, inputs the Kripke structure. The main flow of the model checking algorithm is explained below, based on the components, inputs and outputs within the block diagram in Figure 4.

1. The ambient calculus parser (A) inputs the ambient calculus specification (a) and outputs the parsed specification to the state generator (C).
2. The ambient logic parser (B) inputs the ambient logic formula (b), defines atomic propositions based on spatial ambient logic formula properties, and generates atomic propositions with spatial formulas (d). The association between atomic propositions and spatial modalities is established.
3. The ambient logic parser (B) replaces spatial modalities with atomic propositions and transforms the ambient logic formulas to temporal logic formulas in CTL (c).
4. The state generator (C) generates a state transition system (f) corresponding to the ambient calculus specification (a). The initial state is derived from a given ambient calculus specification. Reduction relations are applied to available capabilities to generate future states. The new states are then added to the STS and the transition relation is calculated.
5. The spatial model checker (D) generates the Kripke structure (e) from the state transition system (f) and atomic propositions with spatial formulas (d). Kripke structure calculation involves the state labeling operation. A state label reflects the values of atomic propositions in that state. Each state is associated with an ambient topology. Spatial model checking is applied to the spatial formula with respect to the current ambient topology. New states are added into the Kripke structure together with their labels (i.e. values of atomic propositions).
6. The temporal model checker (E) generates NuSMV code. The NuSMV code is derived from the CTL formula (c) and the Kripke structure (e). The NuSMV model checker is then executed and results and counter-examples (g) are generated.

4.4. Graphs for representing process specifications and logic formulas

In our proposed method, graphs are used to represent process specifications and logic formulas. A process specification is associated with state information, which includes the static and dynamic properties of a process.

Ambient topology represents the locations and the hierarchical organization of ambients. Ambient topology defines the static aspects of a state. Ambient calculus capabilities and their interdependence define the dynamic aspects of a state. Distinct data structures are used to keep the static and dynamic aspects of a state. In contrast to our approach, Mardare et al. [22] represent the state information using sets; Charatonik et al. [20, 21] represent calculus and logic information as strings and their model checking algorithm is a series of operations on strings.

Definition 11 *Ambient topology is defined as an acyclic digraph* $G_{AT} = (N_{AT}, A_{AT})$. $v \in N_{AT}$ *is the set of nodes in which the elements denote ambients. The set of ambients within an ambient calculus specification is denoted by* Λ. *Arcs in the ambient topology are defined as* $a \in A_{AT}$, $a = \{xy \mid x, y \in N_{AT}\}$. *The arcs denote parent-child relations of ambients. For a node* v, *the indegree is defined as* $deg^-(v) = 1$ *and the outdegree is defined as* $deg^+(v) \in \mathbb{N}$.

The *capability tree* is a novel data structure defined in the proposed algorithm. The definition is as follows.

Definition 12 *A capability tree is defined as an acyclic digraph* $G_{CT} = (N_{CT}, A_{CT})$. *The set of nodes in the capability tree* $v \in N_{CT}$ *denotes ambient calculus capabilities. The set of arcs in the capability tree* $a \in A_{CT}$, $a = \{xy \mid x,\ y\ \in\ N_{CT}\}$ *determines the priority among ambient calculus capabilities. The association of capabilities to ambients as well as their effected ambients are determined by the information within the nodes. For any node* v *the incoming degree is equal to 1, i.e.* $deg^-(v) = 1$. *The outgoing degree is calculated as* $deg^+(v) \in \mathbb{N}$.

The acyclic digraph G_F represents an ambient logic formula, in which the nodes represent logical connectives as well as locations. The arcs represent the relation between the operator and the operand. Ambient logic connectives may have different structures. Because of this property, a formula graph may contain nodes and arcs of various types.

Definition 13 *An ambient logic formula is defined as an acyclic digraph* $G_F = (N_F, A_F)$, *where:*

- $N_F = (N_L \cup N_{Binary} \cup N_{Unary} \cup N_{PC})$ is the set of nodes. The set of nodes N_L represents ambients. The labels of N_L are defined by the set Λ. Unary connectives ($\neg$, $\diamond$, $\diamondsuit$) are represented by N_{Unary}. The binary connective ($\vee$) is represented by N_{Binary}. Parallel compositions are represented by N_{PC}.
- The set of arcs is defined as $A_F = (A_{PC} \cup A_{Binary} \cup A_{Unary})$. Parallel compositions are represented by the set A_{PC}. Binary connectives are represented by the set A_{Binary}. Unary connectives are represented by the set A_{Unary}.
- $a_{pc} \in A_{PC} = x, y | x \in N_{PC},\ y \in (N_L \cup N_{Binary} \cup N_{Unary}),\ a_u \in A_{Unary} = (x, y \mid x \in (N_L \cup N_{Unary}), y \in N_{PC})$, $a_b \in A_{Binary} = (x, y | x \in N_{Binary},\ y \in N_{PC})$.
- $\forall v \in N_F, deg^-(v) = 1,\ \forall v \in N_{PC}, deg^+(v) \in \mathbb{N},\ \forall v \in N_{Unary}, \forall v \in N_L, deg^+(v) = 1,\ \forall v \in N_{Binary}, deg^+(v) = 2$.
- A special attribute is used to model the **T** construct of ambient logic. If a node in the set N_{PC} has a **T** attribute with true value, this indicates that the parallel composition includes the constant **T**.

An example ambient topology and capability tree are presented in Figure 5. These structures make up the state information.

4.5. Formula reduction

To be able to use an existing temporal model checker, the ambient logic formulas have to be reduced to temporal logic formulas. Temporal operators of ambient logic are *Sometime* ($\diamondsuit$) and *Everytime* ($\Box$) connectives. These operators are equivalent to EF and AG operators of CTL, respectively. For reducing ambient logic formulas into a CTL equivalent form, the spatial modalities of ambient logic formulas are converted to atomic propositions.

Since we utilize an existing temporal model checker, some of the ambient logic formulas should be restricted, such as $n[\diamondsuit\mathscr{A}]$, $n[\Box\mathscr{A}]$, $\diamondsuit\mathscr{A}|\diamondsuit\mathscr{B}$, $\Box\mathscr{A}|\Box\mathscr{B}$, $\diamond\Box\mathscr{A}$, and $\diamond\diamondsuit\mathscr{A}$. Such ambient logic formulas are not reducible to CTL formulas because they have temporal operators as subformulas of spatial operators. In the

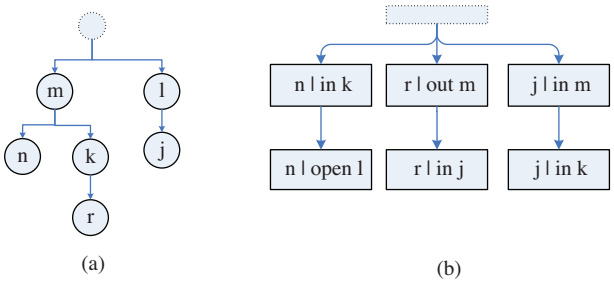

Figure 5. Internal representation of state information. Graph (a) is the ambient topology of state and graph (b) is capability tree.

proposed algorithm, ambient logic formulas must be restricted to temporal operators at higher levels compared to spatial operators. The restriction also brings the advantage that other CTL operators like AF or EG may be used in a more straightforward way.

4.6. STS generation

The initial specification of the model is used to generate the STS. In order to generate states, the capabilities within the ambient calculus specification are executed. The specifications may not include the replication construct. As a result of this property, an acyclic digraph can represent a STS. In this graph, states are represented by nodes and capability executions are represented by edges. There may be parallel capabilities that are possible to be executed at one node. As a result the transitions within the STS are branching and not linear.

A capability is selected at each step for execution. At this step, some condition tests are required whenever a capability will be executed. An object ambient is an ambient upon which a capability is executed. A subject ambient is an ambient that executes a capability. The test conditions depend on the object ambient location and whether a subject ambient is available. If the object ambient does not reside in the current location, a capability with the name of this ambient may not be executed. Similarly, a capability may not be executed if the subject ambient, or its parent, has a prefix capability path (a capability path that prefixes other capabilities).

In Figure 6, an example STS is presented for the ambient calculus process specification $P = m[in\ k.open\ l.n[]|k[out\ m.in\ j.r[]]]|l[in\ m.in\ k.j[]]$. The edges represent ambient calculus capabilities that cause a transition.

A novel data structure called a *capability tree* is proposed for representing temporal aspects. The availability of a subject ambient does not need to be checked when using this data structure. The interdependence of capabilities are represented by acyclic digraphs, which denote capability paths. Capability paths are sequences of capabilities. Capability trees are constructed at the parsing step, which eliminates the need for preprocessing. A capability is selected starting from the root node. The algorithm guarantees that execution of capabilities for parent processes takes place before those of child processes.

4.7. Evaluation of spatial connectives

The satisfaction relation needs to be formally expressed and evaluated in order to build a model checker for ambient calculus and ambient logic. At each generated state the satisfaction relation is evaluated. Formula reduction is applied to ambient logic formulas in order to decompose them into a set of temporal and spatial

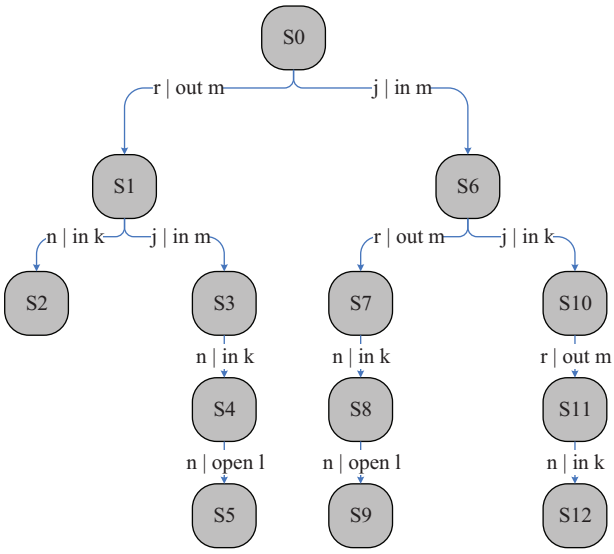

Figure 6. An example STS corresponding to an ambient calculus process.

formulas. The spatial model checking algorithm inputs graphs representing the ambient topology and spatial formulas. Before generation of Kripke structures, the spatial model checking algorithm is applied.

Each node in the ambient topology graph is assigned to a node in the spatial formula graph. This is a recursive procedure for matching ambients to the spatial formula. In this procedure, first the root node in the graph corresponding to an ambient topology is assigned to the root node in the graph corresponding to a spatial formula. The assigned nodes may be forwarded as a partial or complete formula to the children nodes. The matching process continues in a recursive manner until all the nodes within the ambient topology are matched to nodes within the spatial formula. For each type of spatial connective, a different match process is applied. Auxiliary heuristic functions are used for reducing the number of trials for applying the match process. Below, these functions and the match process are explained in detail.

4.7.1. Heuristic functions

The spatial connectives of *Parallel Composition* ($|$) and *Somewhere* ($\diamond$) require a search procedure for matching them to ambients in the ambient topology. We introduce heuristic functions in order to increase the performance of this search procedure. In former studies, this search procedure was based on brute force, and every alternative was evaluated within the match process. We propose to reduce the number of trials by using auxiliary heuristic functions. We also introduce the concept of wildcard connectives. These connectives may match different types of ambients representing various logical constructs. The true connective **T** matches any assigned ambient topology. Matching the *Negation* connective to an ambient topology requires that none of its subformulas match that ambient topology. *Somewhere* is another wildcard connective. When matching this connective, the parallel processes of the parent ambient are not processed. Consecutive applications of the $\downarrow$ operator generate a wildcard property for the associated *Somewhere* connective. The *wildcard* function determines whether a formula graph node has the wildcard property. Below we present the pseudocode for this function.

Listing 1. Pseudocode of wildcard heuristic function.

```
PROCEDURE wildcard
 BEGIN
 input: NODE is a formula graph node
 output: boolean constant
 IF NODE is a location return false
 IF NODE is a disjunction
  return wildcard(node.first_child) V wildcard(node.first_child)
 IF NODE is a somewhere return true
 IF NODE is a negation return true
 IF NODE is a parallel_composition
 BEGIN
  IF NODE has a T property
   return true
   FOR EACH child of NODE
   BEGIN
    IF wildcard(child) = true
    return true;
   END
 END
 return false
END
```

When matching the *Disjunction* and *Somewhere* connectives, the expected ambients in the subformulas are not directly derivable. We define a function named *guessExpectedAmbients* at this step. The set of expected ambients corresponding to a formula graph node is returned by this recursive function. The set that is returned by this function includes all possible ambient combinations, which are expected to be reachable from the children nodes. The pseudocode of the *guessExpectedAmbients* function is given below.

Listing 2. Pseudocode of *guessExpectedAmbients* heuristic function.

```
PROCEDURE guessExpectedAmbients
 BEGIN
 input: NODE is spatial formula node
 output: set of set of string
 IF NODE is a location return { {NODE.name }}
 IF NODE is a disjunction return guessExpectedAmbients(node.first_child)
 IF NODE is a somewhere return guessExpectedAmbients(node.child)
 IF NODE is a negation return {{}}
 IF NODE is a parallel_composition return cartesian products of the elements of
  returned values of guessExpectedAmbients for each child
END
```

The parent of an ambient within the ambient topology is given by the recursive function *findSublocation*. The pseducode of the *findSublocation* function is shown below.

Listing 3. Pseudocode of *findSublocation* heuristic function.

```
PROCEDURE findSublocation
 BEGIN
 input: WANTED is string, ROOT is ambient topology node
 output: RESPONSE is ambient topology node
 FOREACH child of ROOT
 BEGIN
  IF child.name = WANTED return ROOT
 END
 FOREACH child of ROOT
 BEGIN
  RESPONSE = findSublocation(WANTED, child)
  IF RESPONSE .NOT. == null return RESPONSE
 END
 return null
END
```

4.7.2. Matching of spatial formula

In the next step, the nodes in the ambient topology are matched with nodes in the spatial formula graph. In this procedure, some ambient topology nodes are matched directly with spatial formula graph nodes. Others are forwarded to children nodes in the spatial formula graph. Alternative assignments are tried for nodes in an ambient topology with respect to a spatial formula. In spatial model checking, the matching operation begins from the root node in the graph and progresses recursively down toward the leaf nodes. If a part of a formula is matched to a node, its children are searched for a match to the remaining part of the formula. The semantics of the ambient calculus connectives corresponding to the spatial graph nodes determine the flow of the matching operation.

The matching process at the nodes representing locations is according to Definition 5. The location nodes can be assigned to only one ambient topology node. If the ambient topology node does not have the same name as the location node, the match process for the location node fails. If the ambient topology node has the same name as the location node, the location node assigns the children of the ambient topology node to its parallel composition child. The result of the match is successful if the parallel composition child of the location node succeeds in finding a match between its children and the children of the assigned ambient topology node.

The matching process at the nodes representing the *Negation* connective is according to Definition 2. These nodes can be assigned to a collection of ambient topology nodes. Negation assigns the whole set of the assigned ambient topology nodes directly to its child parallel composition node. If the parallel composition child of the negation node succeeds in finding a match, the match process for the negation node fails. If no match between ambient topology nodes and children of the parallel composition is found, the match process for the negation node is successful.

The matching process at the nodes representing the *Disjunction* connective is according to Definition 3. These nodes can be assigned to a collection of ambient topology nodes. Nodes of type disjunction have 2 parallel composition children. Disjunction assigns the whole set of the ambient topology nodes directly to its child parallel composition nodes. If at least one of the parallel composition nodes succeeds in finding a match, the match process for the disjunction node is successful.

The matching process at the nodes representing the *Somewhere* connective is according to Definition 9. These nodes can be assigned to a collection of ambient topology nodes. A node of the *Somewhere* connective can start matching its parallel composition node from any level of assigned ambient topology node collection. The

match process of nodes for somewhere connectives tries all possible levels of the ambient topology nodes until there is a successful match. We have presented a heuristic to expedite the matching of nodes for somewhere connectives. Searching a single node in the ambient topology is cheaper than trying a full match at every level. The *findSublocation* function finds the level of ambient topology for which the match process should start for the somewhere connective. This technique eliminates the searches of the levels that do not have any possibility of matching.

The matching process at the nodes representing the *Parallel Composition* connective is according to Definition 6. These nodes can be assigned to a collection of ambient nodes. The match process for parallel composition decomposes the assigned ambient topology node collection into subsets that will be forwarded to the children of the parallel composition node. While there are exponentially many alternative decompositions, the number of alternatives is reduced by applying the *guessExpectedAmbients* and *wildcard* functions.

Decompositions are carried out in 2 phases. In the first phase, the expected ambient topology nodes are assigned to child nodes of the node for the parallel composition connective. By the help of the *guessExpectedAmbients* function, the sets of expected ambient topology nodes, called guess sets, are found for each child of the node for the parallel composition connective. Then, every expected ambient topology node, in the collection assigned to the node for parallel composition, is forwarded to the related child.

Because there are ambient topology nodes that are not expected by any child node, these nodes should be assigned to a child of the parallel composition node with the wildcard property or neglected if the parallel composition node has the T (true) property. In the second phase, children of the parallel composition are evaluated for the wildcard property by the *wildcard* function. After determining the set of children with the wildcard property, unexpected nodes of the ambient topology collection are assigned to the elements of this set.

After assignment of all ambient topology nodes, new matching processes are started for all assigned children. If one of the children fails, the match process for the parallel composition node tries to find another decomposition. The match process succeeds if the match processes of all children succeed for a decomposition. In Figure 7, an example of the match process is shown.

4.8. Kripke structure generation

A STS, in which each state has a label consisting of atomic propositions that are valid in that state, is named as a Kripke structure. The system properties are modeled using atomic propositions.

Definition 14 *A Kripke structure is a 4-tuple;* $M = (S, S_0, R, L)$. S *is a finite set of states,* $S_0 \subseteq S$ *is the set of initial states,* $R \subseteq S \times S$ *is the transition relation, and* $L : S \to 2^{AP}$ *is the labeling function that defines the labels of each state. The labels are atomic propositions that are true in this state. The labels are elements of a nonempty set of atomic propositions,* AP.

The data structure that implements the Kripke structure includes the sets S, S_0 and the relation R. Formula reduction is applied to generate the set of atomic propositions. During the formula reduction phase, the spatial formulas are exchanged with atomic propositions. For each state, spatial model checking is applied to generate the labeling function. The Kripke structure results from the attachment of labels resulting from spatial model checking of the states.

4.9. NuSMV code generation

NuSMV is a symbolic model checker based on CTL (computational tree logic) and BDD (binary decision diagrams). The temporal model checking algorithm is implemented using NuSMV. At the previous step, the

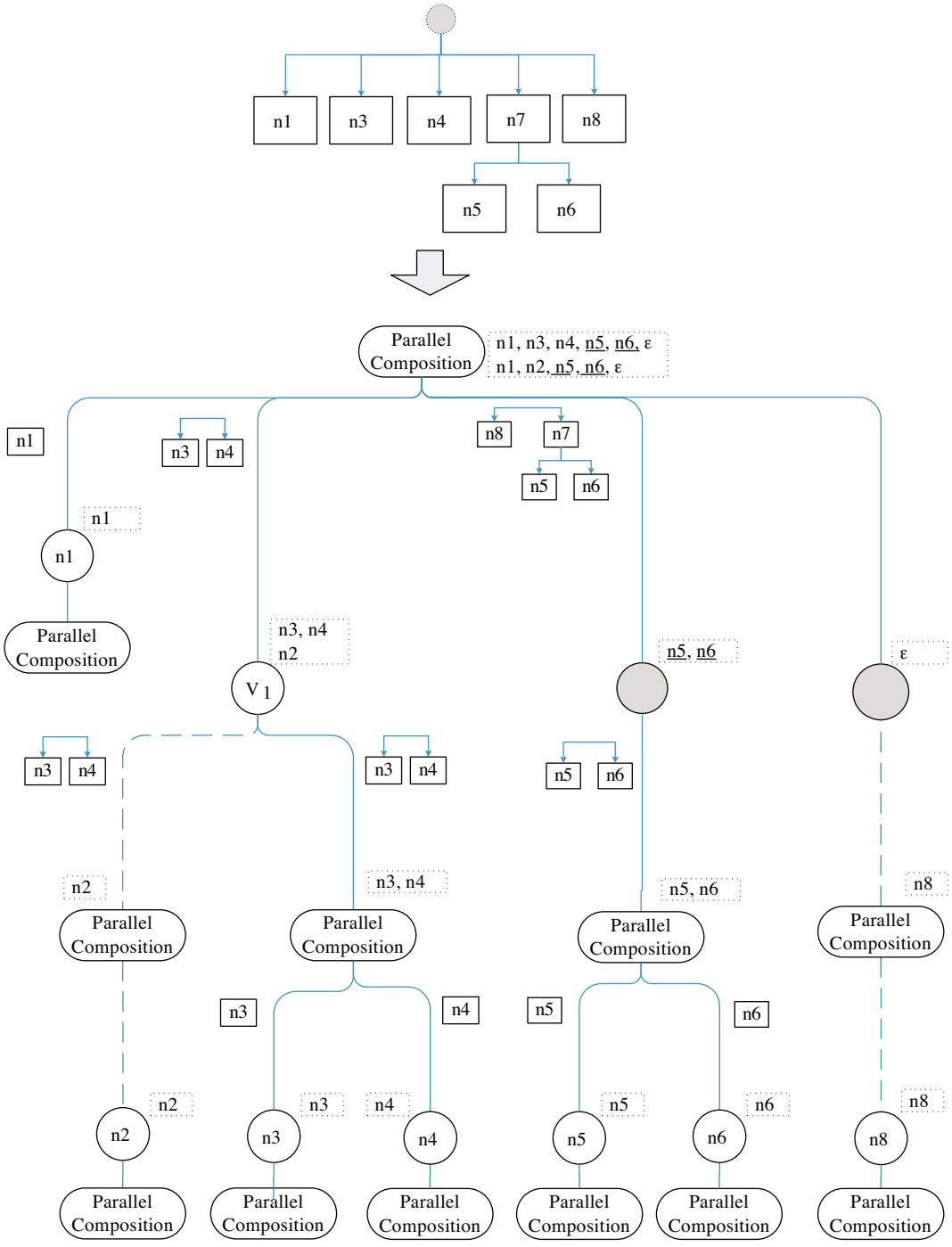

Figure 7. A match example for process P = $n1[] \mid n3[] \mid n4[] \mid n7[n5[] \mid n6[]] \mid n8[]$ and formula F = $n1[] \mid \{n2[] \vee \{n3[] \mid n4[]\}\} \mid \diamond\{n5[] \mid n6[]\} \mid \neg n8[]$. Graphs consisting of rectanglular nodes are ambient topologies assigned to spatial formula graph nodes. Graphs consisting of circular nodes are spatial formula graphs.

formula reduction phase resulted in a Kripke structure associated with a set of CTL formulas. In this step, the Kripke structure and the associated CTL formulas are translated into NuSMV code. A string is generated according to the CTL formula graph. *Sometime* ($\Diamond$) in ambient logic is represented in CTL as the EF operator and *Everytime* ($\Box$) in ambient logic is represented in CTL as the AG operator. Atomic propositions are represented by their names.

In NuSMV, states are defined over state variables with the name *state*. Atomic propositions are represented by boolean variables. A variable is defined in NuSMV for each atomic proposition resulting from the formula reduction. The value of this variable is defined for each state in the Kripke structure. The transitions within the Kripke structure are defined by assignment to the variable *next*.

5. Complexity and performance analysis

In the proposed methodology, the model checking process is decomposed into 2 independent processes as generating STS and checking spatial modalities for states. Time and space complexities for these processes are examined separately.

5.1. Time complexity

Regarding the phase of STS generation, the time complexity is determined by the number of capabilities. When a capability is executed, a single future state is generated. The worst case is that all capabilities are independent. Two capabilities are independent if they exist as a sequence or if their subject ambients are different. In this case, the execution order of the capabilities does not change the set of applicable capabilities. If there are n capabilities, then n capabilities may be executed independently in the first step, $(n-1)$ capabilities may be executed in the second step, and so on until the last capability will be executed in the n_{th} step. The worst case time complexity of generating STS is $O(n!)$. n is the total number of capabilities. The exact formula for the time complexity is given as follows.

$$\sum_{k=0}^{n} \frac{n!}{k!} \tag{1}$$

The type and count of connectives within spatial formulas determine the time complexity for checking spatial modalities. There is a different cost of matching processes for different types of spatial connectives. For the *Location*, *Disjunction*, *Negation*, and *Inactivity* connectives, the search for satisfaction is completed with at most 2 comparisons, while for the *Parallel Composition* ($|$) and *Somewhere* ($\diamond$) connectives, the search requires an arbitrary number of alternatives.

The overall time cost of $\diamond$ connectives, which has child $|$ connectives, is a linear function of the time cost of $|$ connectives. This is calculated as follows. Let a_{ne} be the number of ambients that are at the top level of the specification and are not expected, d_w the number of disjunctions in the $|$ connective with the wildcard property, *not* the number of negation operands associated with the $|$ connective, sw_w the number of $\diamond$ connectives in the $|$ connective with the wildcard property, G the cost of calculating the *guessExpectedAmbients* function, and W the cost of calculating the *wildcard* function.

The match process for $|$ connective consists of 2 phases. The first phase is assignment of expected nodes and the second phase is assignment of unexpected nodes. The cost of the first phase is dependent on the number of disjunction connectives, because each disjunction may cause 2 different expected node combinations. The number of different assignments of expected nodes is given by the following formula.

$$2^{d_w+d} \tag{2}$$

Unexpected nodes of ambient topology can be assigned only to the formula nodes that have a wildcard property. The number of different assignments of unexpected nodes are:

$$a_{ne}^{(sw_w+not+d_w)}. \tag{3}$$

Regarding the match process, the time cost is defined by Eq. (4), which is equal to the product of Eq. (2) and Eq. (3) with addition of the costs of the auxiliary functions:

$$2^{d_w+d} \times a_{ne}^{(sw_w+not+d_w)} + G + W. \tag{4}$$

The time complexity of the match process, corresponding to the calculated time cost in Eq. (4), is given according to the O notation in Eq. (5). The time complexity is exponential. The exponent is proportional to the number of ambients.

$$O(a_{ne}^{(sw_w+not+d_w)}) \tag{5}$$

In contrast, if the the method of brute force search is used, the time complexity becomes as seen in Eq. (6), where $a = a_{ne} + a_e$ represents the total number of ambients at the top level of the ambient topology specification, l represents the number of locations connected by the | connective, sw represents the number of $\diamond$ connectives associated with a | connective without the wildcard property, and d represents the number of disjunctions associated with a | connective without the wildcard property.

$$O((a)^{(sw_w+sw+l+not+d+d_w)}) \tag{6}$$

The time cost of *guessExpectedAmbients* and *wildcard* functions are linear with the number of connectives in the spatial formula because they are called recursively for each connective in the formula. The time cost of finding a match for the $\diamond$ connective is the sum of the cost of the function *findSublocation* and the cost for the | connective. Let PC be the cost of the | connective, which is a child for the $\diamond$ connective; F be the cost of the *findSublocation* function; and a be the total number of ambients. The time cost of the $\diamond$ connective is given with Eq. (7):

$$F + PC. \tag{7}$$

The time cost of brute force search for finding a match for the $\diamond$ connective is given with Eq. (8):

$$a \times PC. \tag{8}$$

In this case, the complexity of the match process and the brute force search is *O(n)*. The time cost of the *findSublocation* function is linear with a because it only looks for an ambient with a specific name in an ambient topology.

The costs of the match processes for the | and $\diamond$ connectives dominate the cost of the spatial model checking process. The overall complexity is reduced by the proposed algorithm. The proposed algorithm and brute force search for match process for the | connective both have exponential complexity, given respectively in Eqs. (5) and (6). Since $(a_{ne} + a_e)^{(sw_w+sw+l+not+d+d_w)} > a_{ne}^{(sw_w+not+d_w)}$, the use of heuristics significantly reduces the cost of the match process. The complexity for the $\diamond$ connective for both the proposed algorithm and brute force search is linear. However, the actual cost is reduced from Eq. (8) to Eq. (7).

5.2. Space complexity

The depth-first calculation method is used to construct the STS. After calculation of its successor states, a state can be discarded from memory. The depth of the STS is limited by the number of capabilities. Regarding the state space generation phase, the space complexity is given by $O(n)$, where n denotes the number of capabilities.

During the overall process in which the spatial modalities are checked, only one copy of a formula is used. The space requirement of this process is equal to the formula size. The formula size depends on the number of logical connectives within the formula. For spatial modalities, the space complexity is given by $O(c)$, where c denotes the number of the connectives within the formula.

5.3. Case study for performance analysis

The case study includes 3 ambient calculus and 2 ambient logic specifications. Ambient calculus specifications are presented in Appendix A and ambient logic specifications are presented in Appendix B. Each ambient calculus specification models a multidomain network where domains, host, user, and files are modeled as ambients. Ambient logic formulas represent different properties of these models.

The case study represents a scenario involving mobility among multiple domains. The case study includes 3 domains, $Domain_A$, $Domain_B$, and $Domain_C$; 4 hosts, $Host_1$, $Host_2$, $Host_3$, and $Host_4$; 3 users, $User_1$, $User_2$, and $User_4$; and 4 files, $File_1$, $File_2$, $File_3$, and $File_4$. This scenario has been captured as ambient calculus specifications *Spec1*, *Spec2*, and *Spec3*. In *Spec1*, $User_1$ does not conduct any action. In *Spec2*, $User_1$ has the rights to login to/logout from $Domain_A$ and $Domain_B$, and to read $File_1$. In *Spec3*, $User_1$ has the rights to logout from $Host_1$, logout from $Domain_A$, and login to $Domain_B$. In all the specifications, $User_2$ has the rights to read and write the files $File_1$, $File_2$, $File_3$, and $File_4$. $User_4$ has the rights to login to/logout from $Domain_B$ and $Domain_C$, login to/logout from $Host_3$, and read $File_3$ and $File_4$. The hosts $Host_2$ and $Host_3$ are accessible to and readable by users of $Domain_C$. $Host_1$ in $Domain_A$ is accessible to $Domain_B$ hosts. Initially, $Host_1$ contains $File_1$, $Host_2$ contains $File_2$, $Host_3$ contains $File_3$, and $Host_4$ contains $File_4$. Initially, $User_1$ is logged in $Host_1$, $User_2$ is logged in $Host_2$, and $User_4$ is logged in $Host_4$.

According to the security policy rule in the case study, $Domain_A$ objects should not be read by $Domain_C$ subjects. The security policy rule has been formalized as *Formula1* and *Formula2* according to detail of specification. *Formula1* states that "The *World* contains $Domain_A$, $Domain_B$, and $Domain_C$. In $Domain_A$, $Host_1$ is connected with any other host; in $Domain_B$, $Host_2$, and $Host_3$ are connected with any other host; in $Domain_C$, $Host_4$ is connected with any other host and after some time, $Host_4$ contains $data_1$ somewhere inside the host". *Formula2* states: "After some time, $Host_4$ contains $data_1$ somewhere inside the host". Both of the formalizations state the result of an unintended information flow where $data_1$, which was originally in $Domain_A$ within $Host_1$, ends up in $Host_4$, which is in $Domain_C$. If one of these formulas is shown to be satisfied by the ambient calculus specifications by the model checker, the meaning is that the unintended information flow takes place and the security policy rule is not satisfied.

The following sequence of actions leads to an unintended information flow: $User_2$ can read $File_1$ and copy or write $File_1$ to $Host_2$ or $Host_3$. This information can be read by $User_4$ of $Domain_C$ by movement into $Domain_B$ and reading from $Host_2$ or $Host_3$. For all the specifications and formulas, the model checker successfully finds the corresponding action sequences where the security policy rule is not satisfied. However, the time and memory cost of the model checking process depends on the properties of alternative formal specifications. This issue is discussed in the next section.

5.4. Performance analysis

A software, which implements the proposed model checking algorithm in the Java language, has been developed. In this section, the implementation is tested with formal specifications of a case study, which is presented in Appendix A and B. The tests were run on a single node of a cluster. The cluster itself has 8 nodes. A node has a CPU of 2.93 GHz speed and 9.76 GB memory. Only one node of the cluster has been utilized since the current version of NuSMV does not support parallel and distributed computations. The model checker application has been deployed as a .jar file and Java 1.6 64-bit edition on a Linux operating system has been used to run the model checker application. NuSMV version 2.5.0 has been used as a temporal model checker.

Table 4. Properties of ambient calculus specifications.

Specification	Number of Ambients	Number of Capabilities	Number of States
Spec1	16	32	560
Spec2	16	39	33,123
Spec3	16	37	628,527

The properties of ambient calculus specifications are shown in Table 4. All the specifications consist of the same set of ambients, where their starting ambient topology is the same. The type of capabilities and the number of capabilities are different between these specifications. Because of this property, the future evolution of the models vary dramatically. Three ambient calculus specifications are evaluated with respect to 2 ambient logic formulas. The time and memory cost for STS is shown in Table 5, for 8 MB and 1 GB heap sizes of the Java Virtual Machine. Since memory management of the Java Virtual Machine includes a garbage collector, the heap size affects performance of Java applications. For this reason, the model checker process has been executed with various heap sizes. The default heap size of the Java Virtual Machine on the cluster was 8 MB. The heap size has been increased to evaluate the effect of heap size on performance. The best performance has been achieved at 1 GB. For the case study, the heap sizes over 1 GB did not result in an increase in performance.

Table 5. STS generation cost.

Specifications	Heap Size	Time (s)	Memory (KB)
Spec1	1 GB	1.039	246,478
	8 MB	1.234	6228
Spec2	1 GB	12.453	350,504
	8 MB	18.120	7734
Spec3	1 GB	154.897	362,384
	8 MB	241.592	7911

In Table 6, the performance results are shown for spatial model checking with a 1 GB heap size of the Java Virtual Machine. The time cost and the space cost for generating the STS is bigger than the cost of spatial model checking. The proposed algorithm provides more significant performance gain for formulas as the number of logical operators increase. In relation to our case study, the proposed model checker was capable of handling a specification with 628,527 states with memory consumption under 8 MB.

We take the following approach for assessing the performance of the proposed algorithm for spatial model checking and comparison to existing work. In order to compare the performance for checking specifications against formulas with different properties, we measure the properties of the 2 given formulas in the case study. The *branching factor* property is related to the number of connectives in the formula. It gives the average number of branches in the syntax tree, which represents a formula. The *depth* property gives the maximum

Table 6. Performance results for spatial model checking.

Specifications	Formulas			
	Formula1		Formula2	
	Time (s)	Memory (KB)	Time (s)	Memory (KB)
Spec1	1.265	262,208	1.520	262,208
Spec2	15.474	350,872	17.134	351,080
Spec3	172.289	362,304	199.815	375,352

depth of the syntax tree, which represents a formula. The properties of given ambient logic formulas in the case study are presented in Table 7.

Table 7. Properties of formulas.

Formula	Branching factor	Depth
Formula1	1	4
Formula2	2.6	3

We compare the use of heuristics against brute force search for exploration of spatial modalities. Since the former studies of Charatonik et al. [20, 21], which use brute force search, do not include an implementation and detailed definitions for procedures, we implemented a variant of our algorithm, which employs brute force search. The results are presented in Table 8. As seen from these results, the proposed algorithm produces better performance results when the branching factor of the formula increases.

Table 8. Performance results for spatial model checking with brute force search.

Specifications	Formulas			
	Formula1		Formula2	
	Time (s)	Memory (KB)	Time (s)	Memory (KB)
Spec1	1.634	262,208	1.829	262,208
Spec2	15.334	350,672	18.405	353,392
Spec3	171.812	364,696	201.765	375,160

The time cost of NuSMV temporal model checking is shown in Table 9. Time cost of NuSMV model checking increases with the number of states. The size of generated NuSMV code grows linearly with the number of states. The specification with 628,527 states could not be processed by NuSMV since it terminated with segmentation fault because of the size of the generated NuSMV code.

Table 9. Performance results of NuSMV with generated code.

Specifications	Formulas	
	Time for Formula1 (s)	Time for Formula2 (s)
Spec1	0.126	0.135
Spec2	463.918	615.361
Spec3	Segmentation fault	Segmentation fault

6. Conclusions and future work

In this study, formal specification of security policies and formal representation of mobility and location constraints have been presented. Our presented approach is built on the formal languages of ambient calculus and ambient logic. The main focus of the specification approach is security policies with mobility and location constraints.

Model checking is applied to verification of security policies with mobility and location constraints. We presented a spatiotemporal model checking algorithm for ambient calculus. We introduced heuristics and novel data structures for reducing the complexity of model checking. An ambient calculus model checker has been implemented and the performance results for the model checker are presented along with a case study.

The results of the performance test showed that the proposed heuristics for checking spatial modalities become faster as the branching factor for the formula increases. A shortcoming of the current implementation of the model checker is the invocation of auxiliary functions. The auxiliary functions are invoked for each spatial model checking process within each state. A more efficient implementation may be developed in which auxiliary functions are invoked only once throughout the spatial model checking process for all states.

The time and space complexity of model checking is largely dependent on the size of the STS. When the number of capabilities increases linearly, the number of states increases exponentially. We aim to apply partial order reduction in order to decrease the number of the states of the STS. This will result in a reduction of time and space cost. We aim to investigate partial order reduction in order to reduce the number of possible orderings derivable from a temporal logic formula. In partial order reduction, parallel interleavings of action sequences are replaced by a single path fragment, which respects the orderings in these sequences. Use of partial order reduction in our algorithm will reduce the number of states corresponding to ambient calculus process specifications that need to be analyzed with respect to ambient logic formulas. Another interesting area for further research is support for parallel and distributed computations for model checking. The current version of the NuSMV temporal model checker does not support parallelism. Utilization of a temporal model checker that supports parallel model checking could provide an increase in performance.

In the current stage, the proposed method is useful for offline verification of security policies with rules of reasonable complexity, which is determined by the number of logical connectives. We have found that the security policy rules in the case study were efficiently handled by the algorithm. However, for very complex policies that have rules with many logical connectives, the analysis will take a long time, which will not be efficient for real-time analysis of actions in the network. For spatial model checking, the number of generated states increases exponentially based on the number of active ambients. Therefore, the proposed algorithm is more suitable for testing of policies in the form of a "what-if" analysis, where the actions and security policy rules for the active elements in the network are introduced to the formal specification in an incremental manner, rather than attempting to represent the entire network with all the security policy rules at one time.

Acknowledgments

The State Planning Organization of Turkey has provided partial support to this project through the TAM Project (2007K120610).

Appendix

A. Ambient calculus specifications for case study

$$Spec1 ::= World[Domain_A[Host_1[User_1[]|File_1[data_1[out\ File_1.0|out\ Host_1.0|out\ Domain_A.0 |in\ Domain_B.0|in\ Domain_C.0|in\ Host_4.0|in\ Host_2.0|out\ Host_2.0|in\ File_3.0|in\ User_2.0 |out\ User_2.0|in\ User_4.0|out\ User_4.0|in\ Host_3.0]]]]|Domain_B[Host_3[File_3[]]|Host_2[User_2 [in\ File_1.0|in\ File_2.0|out\ File_1.0|out\ File_2.0|in\ File_3.0|in\ File_4.0|out\ File_3.0|out\ File_4.0] |File_2[]]]|Domain_C[Host_4[User_4[out\ Domain_C.0|in\ Domain_B.0|in\ Host_3.0|in\ File_3.0| out\ File_3.0|out\ Host_3.0|out\ Domain_B.0|in\ Domain_C.in\ Host_4.in\ File_4.0]|File_4[]]]]$$

$$Spec2 ::= World[Domain_A[Host_1[User_1[out\ Domain_A.0|in\ Domain_A.0|in\ Domain_B.0|out\ Domain_B.0 |in\ File_1.0|out\ File_1.0]|File_1[data1[out\ File_1.0|out\ Host_1.0|out\ Domain_A.0|in\ Domain_B.0 |in\ Domain_C.0|in\ Host_4.0|in\ Host_2.0|out\ Host_2.0|in\ File_3.0|in\ User_1.0|in\ User_2.0| out\ User_2.0|in\ User_4.0|out\ User_4.0|in\ Host_3.0]]]]|Domain_B[Host_3[File_3[]]|Host_2[User_2[in\ File_1.0|in\ File_2.0|out\ File_1.0|out\ File_2.0|in\ File_3.0|in\ File_4.0|out\ File_3.0| out\ File_4.0]|File_2[]]]|Domain_C[Host_4[User_4[out\ Domain_C.0|in\ Domain_B.0|in\ Host_3.0 |in\ File_3.0|out\ File_3.0|out\ Host_3.0|out\ Domain_B.0|in\ Domain_C.in\ Host_4.in\ File_4.0]|File_4[]]]]$$

$$Spec3 ::= World[Domain_A[Host_1[User_1[out\ Host_1.0|out\ Domain_A.0|in\ Domain_B.0]|File_1[data1[out\ File_1.0|out\ Host_1.0|out\ Domain_A.0|in\ Domain_B.0|in\ Domain_C.0|in\ Host_4.0|in\ Host_2.0| out\ Host_2.0|in\ File_3.0|in\ User_1.0|out\ User_1.0|in\ User_2.0|out\ User_2.0|in\ User_4.0|out\ User_4.0| in\ Host_3.0]]]]|Domain_B[Host_3[File_3[]]|Host_2[User_2[in\ File_1.0|in\ File_2.0|out\ File_1.0|out\ File_2.0 |in\ File_3.0|in\ File_4.0|out\ File_3.0|out\ File_4.0]|File_2[]]]|Domain_C[Host_4[User_4[out\ Domain_C.0 |in\ Domain_B.0|in\ Host_3.0|in\ File_3.0|out\ File_3.0|out\ Host_3.0|out\ Domain_B.0|in\ Domain_C. in\ Host_4.in\ File_4.0]|File_4[]]]]$$

B. Ambient logic formulas for case study

$$Formula1 ::= \Box\{\neg\diamondsuit\{\diamond\{Host_4[\diamond\{data_1[T]|T\}]\}\}\}$$

$$Formula2 ::= \Box\{World[Domain_A[Host_1[T]|T]|Domain_B[Host_2[T]|Host_3[T]|T] |Domain_C[Host_4[T]|T]]\} \vee \diamondsuit\{\diamond\{Host_4[\diamond\{data_1[T]\}]|T\}\}$$

References

[1] D. Ünal, O. Akar, M. Ufuk Çağlayan, "Model checking of location and mobility related security policy specifications in ambient calculus", Lecture Notes in Computer Science, Vol. 6258, pp. 155–168, 2010.

[2] M. Becker, C. Fournet, A. Gordon, "Design and semantics of a decentralized authorization language", Proceedings of the 20th IEEE Computer Security Foundations Symposium, pp. 3–15, 2007.

[3] S. Jajodia, P. Samarati, M.L. Sapino, V.S. Subrahmanian, "Flexible support for multiple access control policies", ACM Transactions on Database Systems, Vol. 26, pp. 214–260, 2001.

[4] S. Jajodia, P. Samarati, V.S. Subrahmanian, "A logical language for expressing authorizations", Proceedings of the IEEE Symposium on Security and Privacy, pp. 31–42, 1997.

[5] E. Bertino, F. Buccafurri, E. Ferrari, P. Rullo, "A logical framework for reasoning on data access control policies", Proceedings of the 12th IEEE Computer Security Foundations Workshop, pp. 175–189, 1999.

[6] N. Damianou, N. Dulay, E. Lupu, M. Sloman, "The Ponder policy specification language", Proceedings of the International Workshop on Policies for Distributed Systems and Networks, pp. 18–38, 2001.

[7] T.Y.C. Woo, S.S. Lam, "Authorizations in distributed systems: A new approach", Journal of Computer Security, Vol. 2, pp. 107–136, 1993.

[8] F. Cuppens, C. Saurel, "Specifying a security policy: a case study", Proceedings of the 9th IEEE Computer Security Foundations Workshop, pp. 123–134, 1996.

[9] T. Ryutov, C. Neuman, "Representation and evaluation of security policies for distributed system services", Proceedings of DARPA Information Survivability Conference and Exposition, pp. 172–183, 2000.

[10] A.A. Yavuz, F. Alagoz, E. Anarim, "A new multi-tier adaptive military MANET security protocol using hybrid cryptography and signcryption", Turkish Journal of Electrical Engineering & Computer Sciences, Vol. 18, pp. 1–22, 2010.

[11] N. Zhang, M. Ryan, D.P. Guelev, "Synthesising verified access control systems through model checking", Journal of Computer Security, Vol. 16, pp. 1–61, 2008.

[12] D. Ünal, M.U. Çağlayan, "Theorem proving for modeling and conflict checking of authorization policies", Proceedings of the International Symposium on Computer Networks, 2006.

[13] M. Drouineaud, M. Bortin, P. Torrini, K. Sohr, "A first step towards formal verification of security policy properties for RBAC", Proceedings of the Fourth International Conference on Quality Software, pp. 60–69, 2004.

[14] K. Sohr, M. Drouineaud, G. Ahn, M. Gogolla, "Analyzing and managing role based access control policies", IEEE Transactions on Knowledge and Data Engineering, Vol. 20, pp. 924–939, 2008.

[15] L. Cardelli, A.D. Gordon, "Mobile ambients", Theoretical Computer Science, Vol. 240, pp. 177–213, 2000.

[16] D. Scott, Abstracting application-level security policy for ubiquitous computing, PhD thesis, University of Cambridge, 2005.

[17] A. Compagnoni, P. Bidinger, "Role based access control for boxed ambients", Theoretical Computer Science, Vol. 398, pp. 203–216, 2008.

[18] L. Cardelli, A.D. Gordon, "Anytime, anywhere: modal logics for mobile ambients", Proceedings of the 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 365–377, 2000.

[19] L. Cardelli, A.D. Gordon, "Ambient logic", Mathematical Structures in Computer Science, 2006.

[20] W. Charatonik, A. Gordon, J. Talbot, "Finite-control mobile ambients", Proceedings of the 11th European Symposium on Programming Languages and Systems, pp. 295–313, 2002.

[21] W. Charatonik, S.D. Zilio, A.D. Gordon, S. Mukhopadhyay, J. Talbot, "Model checking mobile ambients", Theoretical Computer Science, Vol. 308, pp. 277–331, 2003.

[22] R. Mardare, C. Priami, P. Quaglia, A. Vagin, "Model checking biological systems described using ambient calculus", Proceedings of the 2004 International Conference on Computational Methods in Systems Biology, pp. 85–103, 2005.

[23] D. Hirschkoff, E. Lozes, D. Sangiorgi, "Separability, expressiveness, and decidability in the ambient logic", Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science, pp. 423–432, 2002.

[24] D. Hirschkoff, E. Lozes, D. Sangiorgi, "On the expressiveness of the ambient logic", Logical Methods in Computer Science, Vol. 2, pp. 1–35, 2006.

[25] D.F. Ferraiolo, R. Sandhu, S. Gavrila, D.R. Kuhn, R. Chandramouli, "Proposed NIST standard for role based access control", ACM Transactions on Information and System Security, Vol. 4, pp. 224–274, 2001.

[26] D. Ünal, M.U. Çağlayan, "XFPM-RBAC: XML based specification language for security policies in multi-domain mobile networks", Security and Communication Networks, 2012. In press.

[27] C.C. Giunchiglia, A. Cimatti, E. Clarke, F. Giunchiglia, M. Roveri, "NUSMV: a new symbolic model checker", International Journal on Software Tools for Technology Transfer, Vol. 2, pp. 410–425, 2000.