FPFM: A FORMAL SPECIFICATION AND VERIFICATION FRAMEWORK FOR SECURITY POLICIES IN MULTI-DOMAIN MOBILE NETWORKS

by

Devrim Ünal

B.S., Control and Computer Engineering, Istanbul Technical University, 1997

M.S., Computer Science, University of Sheffield, 1998

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

Graduate Program in Computer Engineering

Boğaziçi University

2011

ACKNOWLEDGEMENTS

First of all, I would like to thank my lovely wife Anna for her continuous and unique support. I would like to thank my supervisor, Prof. Mehmet Ufuk Çağlayan, for his patience through long years of my Ph.D. study and for his excellent skills in directing my Ph.D. research. I would like to thank my examiners Assoc. Prof. Tuna Tuğcu and Assoc. Prof. Albert Levi, for their useful ideas and comments throughout my presentations and drafting of manuscripts. I acknowledge the ideas of my colleagues in the department, specially Burak Gürdağ, Şerif Bahtiyar, Engin Deveci and Murat Cihan for fruitful discussions in the areas of formal methods and security. I also thank Ersin Evin, Aydın Kubilay, Alparslan Babaoğlu and Önder Yetiş, managers and directors in TUBITAK UEKAE and TUBITAK BILGEM during my study, for supporting Ph.D. studies of researchers. I appreciate the support of the State Planning Organization of Turkey, TAM Project (2007K120610). Finally, I would like to thank my family for supporting me through my entire student life.

ABSTRACT

FPFM: A FORMAL SPECIFICATION AND VERIFICATION FRAMEWORK FOR SECURITY POLICIES IN MULTI-DOMAIN MOBILE NETWORKS

We present a framework called Formal Policy Framework for Mobility (FPFM) for the specification and verification of domain and inter-domain security policies in a multi-domain mobile network environment. FPFM supports the specification of security policies with mobility and location constraints, role hierarchy mapping, inter-domain services, inter-domain access rights and separation of duty. The specification of security policies in FPFM is based on a formal security policy model, called FPM-RBAC (Formal Policy Model for Mobility with Role Based Access Control) and a XML based security policy specification language called XFPM-RBAC (XML Based Formal Policy Language for Mobility with Role Based Access Control). Formal verification of security policies ensure that the security policy is satisfied by the network elements in a given network configuration. FPFM supports extraction of formal specifications from defined network configurations, domain and inter-domain security policies. Another novel aspect of FPFM is the support for formal information flow analysis related to mobility within multiple security domains. Automated verification of formal specifications are carried out through model checking and theorem proving. A spatio-temporal model checking algorithm has been proposed and a model checking tool has been developed for spatio-temporal model checking of location and mobility constraints in security policy rules. Conflicts within security policy rules are resolved through theorem proving with the help of the Coq interactive theorem prover.

ÖZET

FPFM: ÇOK ETKİ ALANLI GEZGİN AĞLARDA GÜVENLİK POLİTİKALARI BETİMLEME VE DOĞRULAMA ÇERÇEVESİ

Çok etki alanlı ağlarda gezgin bilgisayarların, kaynakların ve kullanıcıların hareketliliği güvenlik açısından zorluklar meydana getirmektedir. Güvenlik etki alanları arasında hareketlilik içeren eylemler, etki alanında ve etki alanları arasında oluşturulmuş güvenlik politikaları bağlamında betimlenmeli ve doğrulanmalıdır. Bu tez kapsamında, FPFM (Gezginlik için Formal Güvenlik Politikası Çerçevesi) adında, çok etki alanlı gezgin ağlarda kullanıma yönelik, bir güvenlik politikası betimleme ve doğrulama çerçevesi ortaya konulmaktadır. FPFM, gezginlik ve konum kısıtları, rol hiyerarşileri eşleştirme, etki alanları arası servisler, etki alanları arası erişim hakları ve görevlerin ayrımı unsurlarını içeren güvenlik politikalarının betimlenmesini desteklemektedir. Güvenlik politikalarının betimlenmesi için FPM-RBAC adı verilen bir formal güvenlik politikası modeli ve XFPM-RBAC adı verilen XML tabanlı bir güvenlik politikası dili önerilmektedir. Güvenlik politikalarının doğrulanması, belirli bir ağ yapılandırması içerisinde, güvenlik politikalarının sağlandığının onaylanmasını sağlar. FPFM bu kapsamda tanımlı ağ yapılandırması, etki alanı güvenlik politikası ve etki alanları arası güvenlik politikasından formal betimlemelerin üretilmesini sağlamaktadır. FPFM'in katkı sağladığı alanlardan bir başkası, birden fazla etki alanı içerisindeki gezginlik kaynaklı bilgi akışlarının formal analizidir. Formal betimlemelerin otomatik doğrulanması için model denetleme ve teorem ispatlama yöntemleri kullanılmaktadır. Güvenlik politikaları içerisindeki konum ve hareketlilik kısıtlarının uzay-zaman tabanlı model denetlemesi için bir uzay-zaman tabanlı algoritma önerilmiş ve bir model denetleme aracı geliştirilmiştir. Coq etkileşimli teorem doğrulama aracı kullanılarak güvenlik politikaları içerisindeki çelişkilerin çözümlenmesi sağlanmıştır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

$in \quad M$	Ambient Calculus Enter M
n	Ambient Calculus Name
x	Ambient Calculus Variable
$(x).P$	Ambient Calculus Input
$open \quad M$	Ambient Calculus Open M
$out \quad M$	Ambient Calculus Exit M
$\mathscr{A}, \mathscr{B}, \mathscr{C}$	Ambient Logic Expressions
$\mathscr{A} \mid \mathscr{B}$	Ambient Logic Composition
$\neg \mathscr{A}$	Ambient Logic Negation
$\mathscr{A} \vee \mathscr{B}$	Ambient Logic Disjunction
$n[\mathscr{A}]$	Ambient Logic Location
A	FPFM-RBAC Set of Actions
ACT	FPFM-RBAC Set of Signed Actions
ADM	FPFM-RBAC Administrator predicate
ADU	FPFM-RBAC ActiveDomainUser predicate
AO	FPFM-RBAC Set of Authorization Objects
AS	FPFM-RBAC Set of Authorization Subjects
AT	FPFM-RBAC Set of Authorization Terms
AT_Γ	FPFM-RBAC Set of Inter-Domain Authorization Terms
C	FPFM-RBAC Set of Constraints
$\mathcal{D}$	FPFM-RBAC Security Domain
DR	FPFM-RBAC DescendantRole predicate
EDH	FPFM-RBAC EnrolledDomainHost predicate
EDR	FPFM-RBAC EnrolledDomainUser predicate
H	FPFM-RBAC Set of Hosts
HD	FPFM-RBAC Relation to Map Hosts to Domains
$\mathcal{I}_\Gamma$	FPFM-RBAC Set of Inter-Domain Services
M	Ambient Calculus Capabilities
$\langle M \rangle$	Ambient Calculus Asynchronous output

$M.M$	Ambient Calculus Path
$M[P]$	Ambient Calculus Ambient
$M.P$	Ambient Calculus Capability
N	FPFM-RBAC Set of Signs Representing Permission or Denial
O	FPFM-RBAC Set of Objects
OIT	FPFM-RBAC ObjectIsType predicate
OTH	FPFM-RBAC Object Type Hierarchy
P, Q	Ambient Calculus Processes
$P\|Q$	Ambient Calculus Composition
$\mathcal{P}$	FPFM-RBAC Security Policy
PA	FPFM-RBAC Permission Assignment Relation
PA_Γ	FPFM-RBAC Inter-Domain Permission Assignment Relation
R	FPFM-RBAC Set of Roles
R_Γ	FPFM-RBAC Set of Inter-Domain Roles
RAS	FPFM-RBAC RoleAssumed predicate
REN	FPFM-RBAC RoleEnabled predicate
RH	FPFM-RBAC Role Hierarchy
RH_Γ	FPFM-RBAC Inter-Domain Role Hierarchy
RM	FPFM-RBAC Role Map
RM_h	FPFM-RBAC Home Role Map
RM_f	FPFM-RBAC Foreign Role Map
RSG	FPFM-RBAC RoleAssigned predicate
$\mathcal{S}$	FPFM-RBAC Service
$\hat{\mathcal{S}}$	FPFM-RBAC Inter-Domain Service
SA	FPFM-RBAC Service Access Relation
SA_Γ	FPFM-RBAC Inter-Domain Service Access Relation
T	FPFM-RBAC Set of Object Types
U	FPFM-RBAC Set of Users
UA	FPFM-RBAC User Assignment Relation
UD	FPFM-RBAC Relation to Map Users to their Home Domains
$\mathcal{V}$	FPFM-RBAC Set of services
$\mathcal{W}_\Gamma$	FPFM-RBAC Inter-Domain Security Policy

γ	FPFM-RBAC Number of Inter-Domain Services
ϵ	Ambient Calculus Null
ζ	FPFM-RBAC Number of Object Types
η	Ambient Logic name n
κ	FPFM-RBAC Number of Hosts
ρ	FPFM-RBAC Number of Roles
σ	FPFM-RBAC Number of Services
ϱ	FPFM-RBAC Number of Inter-Domain Roles
τ	FPFM-RBAC Number of Objects
υ	FPFM-RBAC Number of Users
Γ	FPFM-RBAC Set of Security Domains
Ω	FPFM-RBAC Set of Domain Security Policies
0	Ambient Logic Void, Ambient Calculus Inactivity
$\top$	Ambient Logic Logical True
$\rightarrow$	Ambient Calculus Reduction relation
$\downarrow$	Ambient Calculus Nesting relation
$\diamond$	Ambient Logic Somewhere
$\diamondsuit$	Ambient Logic Sometime
$\Box$	Ambient Logic Everytime
$\models$	Ambient Logic Satisfaction relation

LIST OF ACRONYMS/ABBREVIATIONS

BDD	Binary Decision Diagram
BNF	Backus-Noir Form
CCS	Calculus of Communicating Systems
CIC	Calculus of Inductive Constructions
CS	Set of Conflicting Services
CR	Set of Conflicting Roles
CP	Set of Conflicting Permissions
CSP	Communicating Sequential Processes
CTL	Computational Tree Logic
DCR	Dynamic Separation-of-Duty Based on Roles
DCS	Dynamic Separation-of-Duty Based on Services
SLCR	Dynamic Separation-of-Duty Based on Roles with Location Constraints
SLCS	Dynamic Separation-of-Duty Based on Services with Location Constraints
FPFM	Formal Policy Framework for Mobility
FPM-RBAC	Formal Policy Model for Mobility with Role Based Access Control
GUI	Graphical User Interface
HCPN	Hierarchical Coloured Petri Net
ISA	Interconnection Security Agreement
LCONF	Location Configuration
LTL	Linear Temporal Logic
MT	Mobile Terminal
PAM	Permission Assignment Matrix
RBAC	Role Based Access Control
SAM	Service Access Matrix
SCR	Static Separation-of-Duty Based on Roles
SCS	Static Separation-of-Duty Based on Services
SICR	Static Inter-Domain Separation-of-Duty

SLCP	Static Separation-of-Duty Based on Permissions with Location Constraints
SLCR	Static Separation-of-Duty Based on Roles with Location Constraints
SLCS	Static Separation-of-Duty Based on Services with Location Constraints
SRM	Static Inter-Domain Separation-of-Duty Based on Role Mapping
SOD	Separation-of-Duty
SPMI	Security Policy Management Interface
XACML	Extensible Access Control Markup Language
XFPM-RBAC	XML Based Formal Policy Language for Mobility with Role Based Access Control
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformations
VPN	Virtual Private Network

1. INTRODUCTION

The provision of services in networks with multiple administrative domains requires support for cross-domain security policy specification, enforcement, management and verification. Next generation wireless networks will provide seamless mobility for users to support ubiquitous computing. The proliferation of ubiquitous computing enables users to remain connected to network resources irrespective of their location. The multi-domain mobile network environment consists of multiple interconnected domains and mobile users, hosts and objects as sketched in Figure 1.1. Interconnection and mobility are the two main concepts that come into consideration where users are allowed to use network connectivity of multiple domains. An interconnection is defined as "the direct connection of two or more Information Technology systems for the purpose of sharing data and other information resources" [2].

Interconnections of networks for most commercial, government and military organizations require strict security mechanisms defined by inter-domain security policies. Mobility of users between domains require that users are able to connect to the networks of multiple administrative domains, possibly using a single identity. Inter-domain policies in such an environment need to support concepts such as locations, mobility, role mapping, inter-domain access rights and separation of duty between domains.

Mobility of hosts, objects and users present challenges for security in multi-domain mobile networks. The actions involving mobility across security domains need to be specified and verified with respect to domain and inter-domain policies. We are concerned with formal specification and verification of security policy in an environment where users roam between different administrative domains. This problem may be abstracted by the following question: "Are the actions of mobile users compliant with the security policy for the administrative domains that they move in, and the inter-domain security policy between these domains?"

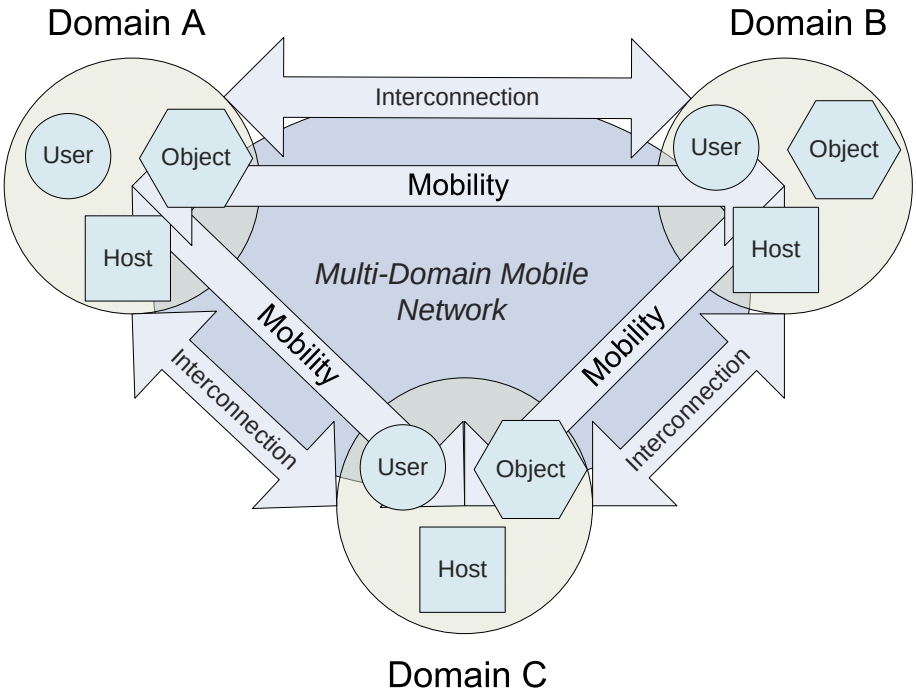

Figure 1.1. Example multi-domain mobile network environment.

A security policy determines the actions that active entities are allowed to conduct on passive entities and defines the conditions. Active entities are also called authorization subjects (or simply subjects) within security policy terminology. Subjects can conduct operations on passive entities called authorization objects (or simply objects). Subjects may be users, roles or hosts. Objects are network resources such as applications, files, databases or messages. Domains and hosts may also behave like passive entities and become authorization objects. Administrative domain defines sets of entities. The actions determine the functions that may be conducted by a subject in an administrative domain. There may be conditions for allowing an action to occur such as time, identity, role membership, user group membership, location and mobility. The rules based on all of these elements constitute the security policies for the administrative domains.

Following these definitions we may formalize our problem definition as follows: "Given a model of a network that includes mobile users roaming different administrative domains with their respective domain security policies and an inter-domain security policy, how to determine formally whether the actions conducted by the users are

compliant with the domain and inter-domain security policies". Our purpose is to verify compliance formally in terms of a formal specification of a security policy with respect to formal representation of location and mobility constraints.

The utilization of a formal security policy framework in a multi-domain mobile network setting allows answering many questions which otherwise would require a manual review process. First of all, we can query a policy base or could answer questions like: Can an operation be conducted? From which location and under which mobility constraints an operation can be conducted? Second, we can verify a situation at any state of the dynamic mobile network model. We can check whether an operation specified in a given network configuration can be safely executed. We can also check that given the permissions in the security policy whether a security vulnerability arises in the network. Third, we can check consistency of a policy specification. We can answer questions such as: Is an action specified as both permitted and prohibited? Are the rules in the domain policy consistent with rules in the inter-domain policy?

In the light of issues presented above, a Formal Policy Framework needs to be able to answer the following questions:

(i) Enforcement*:*
- Is an action executable in accordance with inter-domain policy?
- Is an action allowed within location and mobility constraints?

(ii) Compliance*:*
- Is a set of actions compliant with domain and inter-domain policies under a given network configuration?
- Is domain policy compliant with inter-domain policy, and vice versa?

(iii) Analysis and Verification*:*
- Do a set of policies prevent some known security breaches arising from cross-domain actions and mobility?
- Is the policy set complete/consistent?

Most of the related work provides mechanisms to define security policies but

lack support for answering questions such as those above. Current state-of-the-art in the area of multi-domain security policy specification and verification are mostly related to federated systems [3]. The federated system approach requires a centralized knowledge of all system resources and multi-domain users, which are assumed to be static in the network. This approach is not suitable for multi-domain mobile networks where different administrators exist for different domains and additionally users and resources are mobile. Other studies in the area of role based access control policies with location information is mostly targeted towards location based services in networks, not providing a general model for security policies in a multi-domain mobile network.

In this thesis, we present the Formal Policy Framework for Mobile Networks (FPFM) framework for specification of mobile network configurations and security policies and verification of security policies. To the best of our knowledge, FPFM is the first example of a security policy framework that includes location and mobility constraints, role mapping, inter domain access rights and separation of duty policy rules for multi-domain security policies. The FPFM framework provides integrated support for verification built on a formal security policy model through integrated verification tools. FPFM provides incremental and automated means of developing formal specification of security policies, makes formal methods usable by security administrators and applications, provides tools to specify and verify complex temporal and locational constraints in policies and integrates various formal specification methods and tools in a single framework.

As the formal basis of the FPFM framework, we propose a formal security policy model for the specification of domain and inter-domain security policies in a multi-domain mobile network environment. The presented policy model uses Ambient Calculus, Ambient Logic and Predicate Logic for specification of security policies. Ambient Logic [4] is used to specify dynamic mobility and location constraints in security policy rules. The Ambient Calculus [5] is used to specify the current state of a mobile network and breach scenarios for testing of policies. The matching of mobility and location constraints in policy rules is accomplished by checking the validity of Ambient Logic formulas against Ambient Calculus specifications. Logical constructs based on

Predicate Logic are used for specification of static constraints such as separation of duty.

We also propose a XML-based security policy language named XFPM-RBAC. The XFPM-RBAC language builds upon the formal security policy model FPM-RBAC. XFPM-RBAC provides means to access the formal reasoning environment by system administrators, applications and network elements. A Security Policy Management Interface (SPMI) application is developed for authoring security policies with XFPM-RBAC.

We apply theorem proving and model checking techniques for verification. Coq theorem prover is used for specification of the formal policy model for a given network and for the verification of policies. Each authorization policy is represented in the Calculus of Inductive Constructions (CIC). A model checker for Ambient Calculus has been developed as part of the framework which checks the validity of Ambient Logic formulas against Ambient Calculus specifications.

The first contribution of our framework is the introduction of a formal inter-domain policy model for mobile networks. Another contribution of our framework is the inclusion of a formal mobility model, which is capable of representing mobile network state as well as complex location and mobility constraints in security policy rules. Third contribution is the integration of formal verification tools for model checking and theorem proving into the security policy framework, together with the ability to generate formal specifications from security policies for the purpose of verification. The administration model is distributed, where inter-domain rules are defined for foreign roles acting on home domain resources. Therefore our framework does not require the global knowledge of users and resources and does not introduce conflicts between inter-domain rules of different domains.

With the Role Based Access Control (RBAC) model defined in our framework, the current state-of-the-art in location and context based RBAC models are advanced by introduction of an inter-domain model and mobility constraints in addition to

spatio-temporal constraints. In our work spatio-temporal access control model has the following properties:

- Both subjects and objects are mobile
- Locations are mobile, hierarchical and are based on a formal model that supports spatial model checking.
- Temporal model is based on CTL (Computational Tree Logic)
- The spatio-temporal environment is dynamic and can change with actions in the formal model

Chapter 2 gives background information and summarizes the related work. In Chapter 3, we introduce the concept of Inter-Domain Policies, investigate the need and requirements for a formal policy framework for mobile networks and present the Formal Policy Framework for Mobility (FPFM) components. In Chapter 4, we present FPM-RBAC, the formal security policy model for multi-domain mobile networks. In Chapter 5, we present XFPM-RBAC, XML-based specifications of domain and inter-domain policies with the FPM-RBAC formal security policy model. The approach for formal specification of security policies, formalization of policy rules and the spatio-temporal Ambient Calculus model checker are presented in Chapter 6. Theorem proving approach with the Coq theorem prover is presented in Chapter 7. In Chapter 8 we present two case studies for joint research project and online library inter-domain services for mobile users. Finally, Chapter 9 summarizes our current and future research work.

2. RELATED WORK

In this section, the related work in the area of formal methods for security is covered, with respect to specific topics related to the work presented in this thesis. In Section 2.1, we cover works which are general formal approaches for verification of security of protocols and systems. In Section 2.2, we cover formal methods and languages aimed at specification and verification of security policies. In Section 2.3, we cover formal methods for security in mobile systems. In Section 2.4, we cover recent Role-Based Access Control (RBAC) models which support spatio-temporal constraints. Finally in Section 2.5, we cover general purpose policy languages and access control frameworks.

2.1. Formal Methods for Specification and Verification of Security Properties of Protocols and Systems

The first main group of formal methods are concerned with definition and verification of security properties of a protocol or system. Some of these methods are focused on cryptographic protocols, while some with protection mechanisms such as a firewall.

The Bell La Padula model is used for verifying confidentiality properties, while BAN logic is a means to verify authentication protocols [6]. Integrity models are used to describe what needs to be done to enforce the information integrity policies. The Clark-Wilson model is an application level integrity model [7]. A class of methods as exemplified by [8] uses functional modeling. This study defines a reference model and formal model for firewalls based on this reference model. The reference model focuses on functionality required by firewall systems to enforce network domain security policies. The formalism is based on Hierarchical Colored Petri Nets (HCPN) that are used to express the functionality of mechanisms, to combine them into a system, and to simulate the system in a design tool environment.

Guttman [9] attacks the packet protection problem defined as "how to use the security services provided by router-like services such as packet filtering and the IP security protocols to achieve useful protection in complex networks". Network representation is a undirected bipartite graph,a formal definition of security objectives and use boolean algebra to represent the constraints; the sets permitted at the various filtering points and the consequents of security goals. Binary Decision Diagrams (BDDs) are used to model packet filters, by implementing the boolean algebra of sets solving the abstraction problem.

Many research efforts has focused on methods based on a concept called "strand spaces" for solving the Dolev-Yao problem [10]. This methodology models cryptographic protocols and their security goals, which is used to detect flaws in protocols. These methods are based on simple mathematical modeling notions, such as directed graphs, boolean algebras, and freely generated algebras. This approach has also been extended to address verification of security guarantees of multiple interacting protocols.

A class of methods can be distinguished as model and theory based formalism. Cryptographic protocol verification using the FDR model checker [11], the interactive theorem prover Isabelle [12], and the automatic theorem prover Athena [13] are based on model and theory based formalism.

In [14] the approach of theory checking to analyzing and verifying properties of security protocols is presented. In this approach they generate the entire finite theory of a logic for reasoning about a security protocol. Determination of whether it satisfies a property, is by a membership test. This approach relies on, modeling a finite instance of a protocol in a natural an informal way and placing restrictions on logical inference rules to guarantee that the algorithm terminates. This generates a "finite theory". This approach is implemented by a theory-checker generator.

Theory generation [15] is a new general-purpose technique for performing automated verification. Theory generation is based on automated theorem proving and

symbolic model checking. The basis of this approach is production of a finite representation of a theory, which is all the facts derivable from a set of assumptions. An automated analysis is possible to test for specific properties and make comparison of protocols. They declare to having applied theory generation to more than a dozen security protocols using four different logics of belief confirming flaws discovered earlier.

Hopper [16] reviews two relatively new tools for automated formal analyis of security protocols. One applies the formal methods technique of model checking to the task of protocol analysis, while the other utilizes the method of theory generation, which builds on both model checking and automated theorem proving. For purposes of comparison, the tools are both applied to a suite of sample protocols with known flaws. A heuristic is suggested for combining the two approaches to provide a more complete analysis than either approach can provide alone.

2.2. Formal Languages and Methods for Specification and Verification of Security Policies

A class of works represent security policies as Datalog programs such as [17]. Dougherty [18] extends the Datalog approach by using a combination of relational reasoning and temporal reasoning to model both policies and their environments. They handle two core problems, goal reachability and contextual policy containment. The direction of this work is parallel to ours, for example the goal reachability problem may be implemented using the model checking approach. However use of a first-order relational language such as Datalog restricts the class of policies that can be modeled. Instead of Datalog we utilize Calculus of Inductive Constructions for relational reasoning. Our proposed environment model based on process calculus is more suitable for mobile networks than the state-based approach presented in this work.

We apply theorem proving to verification of security policies. In [19] we use Coq for checking that an authorization security policy is conflict-free, initially and as authorization rules are added and removed, while [20] uses first order linear temporal

logic embedded within Isabelle to formalize and verify RBAC authorization constraints. A more recent study, [21] uses nontemporal and history-based authorization constraints in the Object Constraint Language (OCL) and first-order linear temporal logic (LTL) to verify role-based access control policies with the help of a theorem prover.

Logic-based security policy models provide a general framework for security policy specification. Becker *et al.*'s SECPAL [22] is a formal security policy language for Grid environments. The Flexible Authorization Framework (FAF) is a logic programming based method for definition, derivation and conflict resolution of authorization policies [23, 24]. Another study based on logic that supports explicit denials, hierarchies, policy derivation and conflict resolution is [25]. Ponder [26] is a general purpose formal security policy language. Woo and Lam [27] define a paraconsistent formal language for authorizations based on logical constructs. In [28] deontic logic is used for modeling the concepts of permission, obligation and prohibition with organizational concepts such as responsibility, delegation and time constructs. A security policy language based on the set-and-function formalism is presented in [29].

2.3. Formal Methods for Specification and Verification of Security Policies in Mobile Systems

For specifying dynamic properties of a mobile network and modeling the spatial and temporal effects of actions, we need to use an executable process calculus. Among many types of calculi, Pi calculus [30, 31] and Ambient Calculus [5] are candidates for modeling locations and mobility because they provide constructs for concurrency, communication and names which are capable of representing locations and executable processes. We use the Ambient Calculus because it provides natural means to model location hierarchies and mobility. Ambient calculus, proposed by Cardelli and Gordon, is a process calculus which is able to theorize about concurrent systems that include mobility and locations. Ambient Logic [4, 32] is a modal logic for expressing spatial and temporal properties of Ambient Calculus. Ambient Calculus has been used for modeling and reasoning about security in mobile systems. For model checking of Ambient Calculus specifications, our approach is similar to the work of Mardare *et al.*

[33, 34]. We use a modified version of Mardare algorithm and present an algorithm based on use of capability trees that reduces complexity of state space generation and matching of Ambient Logic formulas to states. Charatonik *et al.* [35] offers exhaustive search for searching possible decompositions of processes and searching sub locations when checking spatial modalities. We offer heuristics for searching possible decompositions of processes and searching sub locations to reduce the search space.

Reasoning about spatial configurations for application level security policies in ubiquitous environments is one of the issues investigated in Scott's PhD thesis [36]. In this study a simplified version of Ambient Calculus and Ambient Logic is used in policy rules of a security policy. BACI_{R} [37] is a boxed Ambient Calculus with RBAC mechanisms used to define access control policies for ambients. Model checking has been applied for verification of security policies. ACPEG [38] is a tool for evaluating and generating access control policies based on first-order logic. In the works [39, 40] model checking is applied to mobile code, where mobile programs are modeled as Labeled Kripke structures and a generic security policy specification language is used to express rules and manipulate locations of the code. Similar to our approach, they support both access control and information flow control specification. We present in [41] a model checker for spatio-temporal model checking of location and mobility related security policy specifications. In our approach, the Ambient Calculus and Ambient Logic [4] are utilized. In contrast to BACI_{R} which places policies inside Ambient Calculus formulas, we use Ambient Calculus for specification of processes and Ambient Logic for specification of policies and complement them with Predicate Logic based relational model. In contrast to Scott's approach, network level policies rather than application level policies will be covered and locations will denote placement in domains and hosts.

2.4. Role Based Access Control Models with Spatio-Temporal Constraints

There are various access control models that include temporal, location and other context-based spatio-temporal constraints. TRBAC [42] and GTRBAC [43] which is an extension of TRBAC are temporal RBAC models. Some of the examples of ac-

cess control models which include location and context-based constraints are, SRBAC [44], LOT-RBAC [45], Spatio-Temporal RBAC [46], STRBAC [47], GEO-RBAC [48], Context Aware RBAC [49] and STARBAC[50] later extended by ESTARBAC [51].

GTRBAC by Joshi *et al.* [43] is a temporal RBAC which extends TRBAC [42] by Bertino *et al.* TRBAC includes role enabling constraints and periodic time statements for temporal constraints. GTRBAC handles the concept of role activation separately from role enabling and adds the concept of constraint enabling. In GTRBAC, temporal constraints may be related to user-role assignment as well as role-permission assignment. Temporal constraints in the GTRBAC model are capable of specifying both peridocitiy and duration constraints, as well as the maximum number of activations for a role during a time period. GTRBAC also includes mechanisms to handle conflicts and SOD in the temporal domain as well as support for role hierarchies. GTRBAC does not include the concept of multiple domains nor support for spatial constraints such as location and mobility.

The SRBAC model proposed by Hansen and Oleschuk [44] adds to the standard RBAC model the ability to specify spatial constraints on enabling and disabling of roles. The set of permissions that a user may activate at a given location may be limited by spatial constraints. SRBAC also includes a Role Hierarchy model which is location dependent i.e. the permission inheritance relationship between roles may also depend on the location of the user which assumes the role. Separation of Duty constraints in SRBAC model are also location dependent such that two roles with assigned permissions may be mutually exclusive for a given location while they are authorized to be activated simultaneously for other locations. This work defines a conceptual model but does not include an architecture for implementing the conceptual model.

STARBAC [50] provides a spatio-temporal condition set, consisting of tuples made by spatial and temporal conditions. Spatial conditions are based on locations and location types, which are elements of simple sets. Temporal conditions are elements of periodic expressions. An example of a spatio-temporal condition in STARBAC is

(XYZ Building, Monday). STARBAC also supports role enabling/disabling through role control commands. An example role control command is <(Office, Officehour), enable'CLERK'>. Space time reasoning in STARBAC is achieved through membership of location mapping and location type mapping relations. The authors extended this model with the name of ESTARBAC [51] and defined access control evaluation algorithms as well as support for Separation-of-Duty (SOD) constraints. Mondal *et al.* [52] present a XML representation of ESTARBAC.

The LoT-RBAC model [45] provides a fine-grained location model based on the concept of logical and physical locations. It supports relations between locations such as *contains*, *overlaps*, *equals*, *meets* or *disjoint*. Definition of location hierarchies as well as location context based on user, role and permission location are possible. Location expressions of the form $l_{expr} = ([ploc_x,], LLOC_y)$ can be used to define location constraints on user, role, permission assignment, role activation and enabling relations of RBAC. Temporal expressions, role enabling/disabling/activation mechanisms as well as the event and trigger constructs are borrowed from the GTRBAC model. Therefore the Lot-RBAC model may be considered as an extension of GTRBAC model for specification of location constraints.

The basic spatio-temporal constructs of the study of Ray *et al.* [46] are similar with LoT-RBAC. This work introduces the notions of temporal and spatial inheritance and activation in role hierarchies and considers the impact of time and location on separation of duty. The model presented adds spatio-temporal constraints to role assignment and permission assignment relations. With regards to the location model, they speak only about a containment relationship. Additionally the location model presented in this study is limited to a simple set of logical locations; location hierarchies and expressions based on locations are not provided.

Kumar *et al.* [47] propose alternative approaches to specification of permissions in the spatio-temporal context. In contrast to Joshi *et al.* [43] they do not use triggers. Instead, the spatio-temporal constraints are attached to a RBAC Access Matrix and are evaluated at run-time. The logical location model of Kumar *et al.* includes spatial

constraints which are combinations of logical locations combined with the operators of $\{\cap, \cup, \setminus\}$. However they do not present any algorithms to evaluate spatio-temporal constraints. They also introduce the notion of *visibility*, which is defined as "specification of whether a set of roles is allowed to know the location and identity of an object". The relation $Vis_L(r, o)$ returns the spatial resolution to apply to request by a user in role r for the position of the object o. However in their work no formal definition of *spatial resolution* is given neither any algorithms to derive the visibility relation is presented.

The GEO-RBAC model proposed by Damiani *et al.* [48] is a Spatial RBAC model based on the concept of spatial role, which represents a geographically bounded organizational function. Boundaries are defined as features, which are characteristics of geometrical objects. The spatial model adopted in this work is in accordance with the Open Geospatial Consortium (OGC) model. Similar to GTRBAC, role enabling and activation are handled separately. A role is said to be *enabled* if the logical position of a user is spatially contained in the extent of that role. The *extent* of a role defines the boundaries in which the role may be assumed by a user. Role schemas are used to define spatial roles, which depend on the extent of roles, logical positions for users contained in the extent and a mapping function from physical locations to logical locations. Concerning role hierarchies, a new concept of schema hierarchies is introduced and the effect of spatial role extents to hierarchies are studied. Additionally Separation-of-Duty constraints are extended to account for the spatial relationships between user location and role extents. The access control decisions in GEO-RBAC are computed through an access-control function which enumerates the set of all enabled roles in a location. GEO-RBAC does not support mobility and multiple domain hierarchies. Additionally the location constraints are limited to the location of users rather than presenting the overall spatial environment.

The studies presented above are more suitable for environments in which the location of users and objects are static in nature. Locations are defined with first-order relations over a set of logical locations. Some formalism is provided to specify complex spatial configurations; but no syntax or semantics are available for mobility of

users or objects. The impact of role assumption, enablement, activation and permission assignment due to mobility are not considered. The LoT-RBAC model includes location hierarchies, however the formal link between the hierarchy model and the location relations are missing. In all these formalisms the spatial model of a system is static and does not represent neither the actions of users nor the mobility of users, objects and locations. Additionally these models are based on a single domain without consideration of inter-domain relations such as role-mapping.

2.5. Policy Languages and Access Control Frameworks

Policies are constraints on the behavior of a system. Policy languages provide means for concrete representation of policies. Languages for expressing constraints on the behavior of systems are declarative rather than procedural. In a declarative language, the constraints are declared without the detail on how to enforce them. Mathematical notations are used for precise representation of policies and reasoning using automated means. First order logic, stratified logic and deontic logic are the most commonly used mathematical notation techniques for policy representation.

Some policy languages are specifically designed for policy systems. Some of the currently available policy languages include XACML (eXtensible Access Control Markup Language) [53] from OASIS and Ponder from Imperial College [26]. This sort of policy languages are generic and do not target multi-domain mobile networks. More specifically, XACML does not have a formal model for role hierarchies, inter-domain security policies, location and mobility constraints and separation of duty constraints. Ponder is more suitable for network services management in an environment where objects and users and their locations are previously known. However in multi-domain mobile environments, objects and users and their locations are not known in advance. Formal verification for general purpose policy languages are considered by various researchers such as [21] and [38], however a formal model to support verification is not part of the specifications of general purpose policy languages.

There are various access control models that extend the Role-Based Access Con-

trol (RBAC) [54, 55] model with temporal, location and other context-based spatio-temporal constraints. GTRBAC [43] is a temporal RBAC model. The GTRBAC model has been extended by several works for multiple domain environments. Piromruen and Joshi [56] present a preliminary work on extension on GTRBAC. In this work some algorithms are presented to extend a local GTRBAC policy to facilitate inter-domain access. For two domains to interoperate, a domain sends access requirements and a set of exported roles to another. The access requirements are fulfilled by amendment of role hierarchy or addition of local roles in the domain which has objects to be accessed. This approach has some drawbacks. First, the role hierarchies in the accessed domain should be modified each time there is a new access requirement. Second, the accessing domain needs to know about the objects on the domain to be accessed in order to create access permissions. If these objects were to be changed or deleted, the accessing domain needs to be notified. Third, there is no shared mapping of roles between domains and the local role hierarchies are exported to other domains. These drawbacks seriously restrict satisfying the least priviledge and need-to-know principles of security. There are other extensions of GTRBAC including [57]. Bhatti *et al.* proposed X-FEDERATE [58], X-GTRBAC [59], X-GTRBAC Admin [60], which are representations of GTRBAC model in XML language.

A XML based policy specification framework which supports location constraints is presented in [61]. In this paper, the location model is based on the GEO-RBAC model [48]. In GEO-RBAC, locations are defined with first-order relations over a set of logical locations. GEO-RBAC model presents a formal relationship between the RBAC and spatial concepts, but the relationship is based on enumeration of all enabled roles in a specific user location. This approach is not suitable for dynamic multi-domain environments. Furthermore, the location aspects focus on the subjects rather than presenting a spatial configuration for the entire system. The limitations of the GEO-RBAC model are discussed in [62]. For example, the authors speak about the need for "the position as well as the policy to be continuously enforced","controlling the mobility" and "the need for modification of spatial roles as the space evolves", which are concepts covered by the FPM-RBAC model which is the formal basis of XFPM-RBAC policy language.

3. FPFM: A FORMAL SPECIFICATION AND VERIFICATION FRAMEWORK FOR SECURITY POLICIES IN MULTI-DOMAIN MOBILE NETWORKS

In this chapter, we present the Formal Policy Framework for Mobility (FPFM). First, we discuss the need and requirements for a formal policy framework for mobile networks. Second, we present the components of FPFM. Third, we explain how security policies are specified in the FPFM framework. Finally, we discuss the methodology in which security policies are verified in the FPFM framework.

3.1. The Need and Requirements for a Formal Policy Framework for Mobile Networks

First, we provide definitions related to security policies in a multi-domain environment. Second, we provide an overview of security vulnerabilities in multi-domain mobile networks, which are introduced by communication and mobility among multiple domains. Third, we introduce the inter-domain security policy concept for the purposes of this thesis. Finally, we discuss possible applications of a framework for security policies in multi-domain mobile networks.

3.1.1. Security Policies in a Multi-Domain Environment

Security policies in a multi-domain environment require specification and verification of multiple policies. Additionally, in a mobile network environment, where users, hosts and objects are mobile, policy rules may become increasingly complex. The heterogeneous nature of mobile networks suggests a common formal policy model. Security breaches in multi-domain mobile networks often arise from insufficient representation and enforcement of multiple actions including mobility. Formal verification of policies by use of a common formal policy model provides means to reduce security breaches arising from incomplete, inconsistent and ambiguous specification of

multi-domain policies.

For the purposes of this study, we use the following definitions for a domain, an inter-domain policy and a service:

Definition 3.1. *A* security administrative domain*, or a* domain *in short, is a logical boundary for an information and communication system governed by a single administrative authority.*

Definition 3.2. Inter-domain policy *defines the access control rules, attributes and mapping for entities and information in multiple domains to be able to securely access or exchange information.*

Definition 3.3. *A* service *is a set of entities and attributes of entities upon which operations are conducted.*

3.1.2. Security Vulnerabilities in Multi-Domain Mobile Networks

The multi-domain mobile network environment presents many challenges to security. The first problem is unintended information flow between domains due to mobility of users. Second, administrative rights for visiting mobile user resources depend on visited domain policies and users and their home domains have no control on these policies. Third, security threat from mobile users may change according to the domains they come from. A mobile visiting user coming from a public network may present more threat than one coming from a trusted network. Therefore a mobile user needs to be given access due to location and mobility constraints.

The movement of hosts may conceal movement of objects. Consider an example in which there are two domains, Domain A and Domain B and an information flow security policy which states that Domain A files should not exist in Domain B. Domain B users are allowed to move between domains and to read files from both domains. This movement could breach the information flow security policy, since a user may read a file in Domain A, move to Domain B and write the file to Domain B.

A method of tracing such movement is by keeping traces of events, but such traces may be hard and time-consuming to analyze once a long time passes in the system. A more optimal approach that is taken here is to keep the current state of the system, that shows the location and movement of objects.

Administrative weakness is inherent in modern operating systems such as Windows and Unix. The user with the root or domain administrator credentials can virtually access any resource within a network or domain. While this may be acceptable for a static world where all users and objects within a network or domain stays within those network or domain, it may cause some serious security problems when inter-domain collaboration and movement is in place. A user's host may carry information not normally open to access by other organizations, but by moving and logging onto a domain, the user's host and the objects on the mobile user's host become accessible to foreign administrators.

Multiple interconnections are also a source of vulnerability for networks. When an organization (A) makes an interconnection with another, the connection mostly conveys a bi-directional information flow. One of the organizations (B) could have an interconnection with another (C); associated with permissions based on the requirements at the time of this interconnection. So-called "back-end interconnection" from the point of view of A, the permissions arising from interconnection of B to C may breach A's security policy.

These examples show that security vulnerabilities may arise if inter-domain security policies are not checked and enforced. Another major problem is mismatch between security policies of multiple organizations. The host and intra-domain security mechanisms are usually setup in a way that all objects and subjects within a domain are known and trusted (employees of an organization, students of a university etc. which have liabilities against their organization). However when inter-domain mobility is allowed, users outside of an organization's responsibility may conduct actions. Those users (and/or their hosts) are subject (or configured) to adhere to their own organization's security policy. This policy may not match the visited institution's

policy. If there is no enforcement of inter-domain security policy within a domain, the overall policy enforcement may be endangered.

3.1.3. Inter-domain Security Policy

An inter-domain security policy is based on a set of security agreements by participating organizations. A basic inter-domain policy structure is given by NIST with "Interconnection Security Agreement (ISA)" document [2]. However the nature of next generation networks require many other issues to be covered by an inter-domain policy. The various components of an inter-domain policy is presented in Figure 3.1. An inter-domain policy for mobile networks should cover the following concepts:

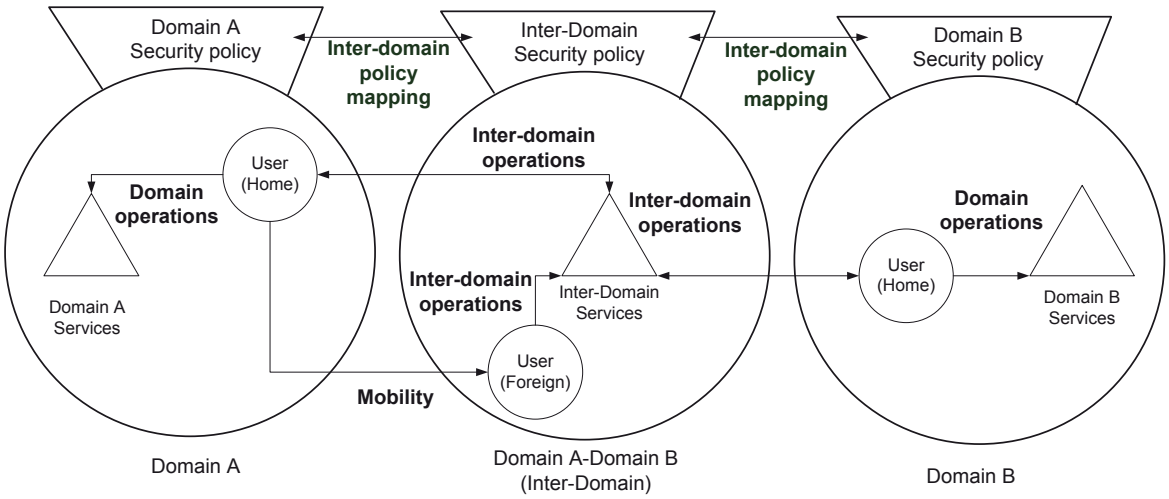

Figure 3.1. Inter-domain operations governed by inter-domain security policies.

(i) Locations: Inter-domain services provided based on locations, such as location based services require support for access control based on locations.

(ii) Mobility: The inter-domain mobility aspects relate to object, host and user mobility across domains.

(iii) Role mapping: This maps the user roles in one domain to another. e.g. Lecturer in one university may become Researcher in another.

(iv) Inter-domain access rights: These rights relate to inter-domain operations. They determine access rights for roles from other domain, access rights for local roles when accessing from another domain and access rights for public access.

(v) Separation of duty: The assignment and assumption of multiple roles to users

should be restricted based on inter-domain mappings and locations in addition to conventional role based separation of duty.

Our study is the first example of a security policy framework to cover all of the above concepts. Former studies do not cover locations, mobility, role mapping, inter-domain access rights and separation of duty as a whole. In our study, we cover location and mobility constraints, role mapping, inter domain access rights and separation of duty policy rules for multi-domain security policies.

Another important difference from related work is our view of the multi-domain environment based on a home domain, foreign domains and inter-domain policies between these domains. Inter-domain policy for each domain is defined by the security administrator of the home domain based on foreign domain roles, eliminating the need for global knowledge of the network users and resources.

An application domain for multiple-domain mobile security policies is roaming between mobile networks of different organizations. As an example, consider a user with a mobile internet subscription visiting a university with a higher speed wireless local area connection. If the inter-domain agreements between the visited university and the mobile internet provider permits, based on location the visiting user may have access to some local resources, such as temporary access to the university library. It is not practical to provide an account from the university domain for a temporary visiting user. The user needs to be recognized by the home domain identity which is the mobile subscriber identity. We assume that the cross-domain authentication mechanisms are in place for identification. The system then needs to map the identity to an inter-domain role. The system also needs to make sure that only intended resources are accessed by the visiting user with an inter-domain role.

3.1.4. Applications of Security Policy Framework

The need for a framework for security policies in multi-domain mobile networks is increasing due to advances in mobile telecommunications, smart mobile comput-

ing devices, service oriented computing and the increasing need for collaboration and information sharing within this new environment.

The provision of mobile inter-domain information services require specification, enforcement and verification of security policy rules which should support specification of role mapping, location, mobility, inter-domain and information flow conditions and constraints. The policy framework presented in this study supports the specification, enforcement and verification of such policies, whereas previous works do not support these concepts as a whole. Here present some scenarios in which the proposed security policy framework may be utilized.

3.1.4.1. On-line Courses. As an example, consider an open university with on-line courses in which various departments and universities are involved. Lecturers give courses on multiple campuses as visiting lecturers, and students may follow these courses from any campus with their mobile terminals. Examinations are only provided within the department that offers the course. Multi-domain access is needed since the lecturers should have access to materials, exam questions and grades for courses which are provided by multiple departments and universities. Information flow should be restricted since grades and exam questions should be available to the lecturer which presents the course. The lecturer's information on his/her own mobile terminal needs to be protected from unauthorized access by foreign domains. The on-line course is a mobile inter-domain service which spans multiple domains of departments and universities, application servers and file servers which host the on-line education web services as well as courses, mobile terminals of lecturers and students as well as the information related to courses such as grades, exam questions and course material.

3.1.4.2. Joint Research Projects. Second, these universities may also be involved in joint research projects with other government, industrial or academic partners. University members such as lecturers and research assistants may assume roles such as research project member or research project coordinator. Roles in each individual domain may be mapped to another through role mapping relations. For example, the

Table 3.1. Example security policy rules for on-line courses.

Rule Type	**Policy Rule**
Role Mapping	Lecturers from University A who are giving courses in on-line university will become visiting lecturer in University B.
Mobility	On-line students may access on-line university courses from any university campus through mobile terminals.
Location	On-line students may take course exams within the department providing the course.
Inter-domain	Lecturers in University A may write grades for on-line courses in University B.
Information Flow	Course grades of a student should not be available to other students.

role of systems analyst in an industrial partner may be mapped to the role of research project member in a university during the project. When the systems analyst visits the university to conduct studies related the project with a mobile computing device, the project files in the research laboratory should be available, while access to other research projects should be restrained. Some information from government partners will be shared to the partners however at the same time some of this information may not be available within the university campus because of physical security concerns. In this case university members may have access to this information through their mobile terminals within government premises. The university members are not allowed to move and copy information from within the government domain to their own domain.

3.1.4.3. Military Networks with Multiple Security Classifications. Third example is inter-connection of military networks with security classification to public networks. As presented in Figure 3.2, in this scenario there are restricted domains, a public domain (such as Internet), mobile terminals (MTs) of users of the restricted domain and public clients/servers that communicate with the restricted domain. MTs use the public network to access restricted information over a virtual private network (VPN). Restricted domains also form a VPN for communication of Restricted information over

Table 3.2. Example security policy rules for joint research projects.

Rule Type	**Policy Rule**
Role Mapping	System Analyst in Industrial Partner domain becomes Research Project Member in Joint Project inter-domain service.
Mobility	System Analyst may connect to University Research Lab with his mobile terminal.
Location	University research assistants may access research project files in the government domain only from within government premises.
Inter-domain	Users from Industrial domain should only access the joint research project files in the University domain.
Information Flow	Files in the government domain may not be written to the university domain.

the Internet. In such a setting, information flows may carry two distinct classification levels, *Restricted* for information for internal use and *Unclassified* for publicly available information.

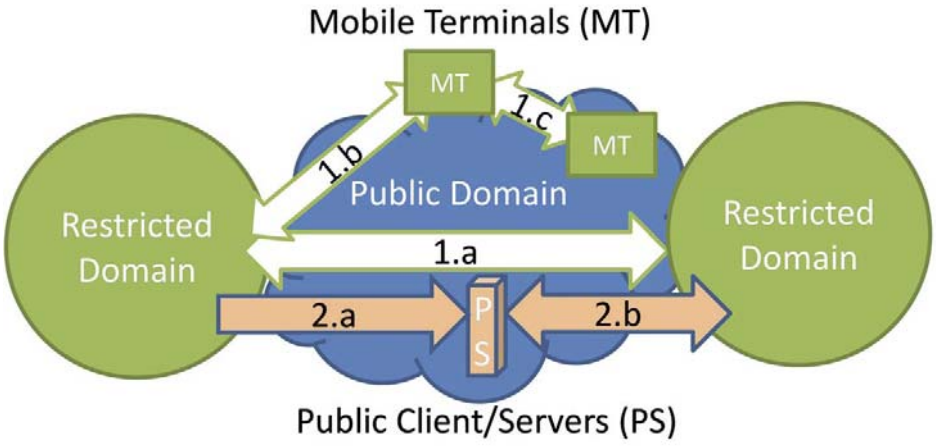

Figure 3.2. Multi-domain scenario with multiple classification levels in military context.

The restricted information flows *1.a, 1.b and 1.c* take place: *a.* between restricted domains, *b.* from MT to Restricted domain or *c.* MT to MT, while the unclassified information flows take place between clients of Restricted domain and clients/servers of Public domain. The information flow *2.a* represents the uni-directional release of public information to Internet, while *2.b* represents bi-directional communication of restricted

network users with public users (such as voice-over-IP). The separation of Unclassified information from Restricted information and therefore preventing the leak of Restricted information to Public domain is the main concern of such a scenario. We list sample security policy rules for military networks with multiple security classifications in Table 3.3.

Table 3.3. Security policy rules for inter-connection of military networks to Internet.

Rule Type	**Policy Rule**
Role Mapping	Restricted network user becomes a Remote access user when logged into a mobile terminal over the public network.
Mobility	Access to restricted information from mobile terminals is permitted only for ranks Captain and above.
Location	For all ranks unclassified information may be transmitted over the public network with mobile terminals.
Inter-domain	Only unclassified information may be released from Restricted domain to servers in the Public domain.
Information Flow	Information flows carrying Restricted information may not terminate on the servers and clients of the Public domain.

3.2. Formal Policy Framework for Mobility (FPFM) Components

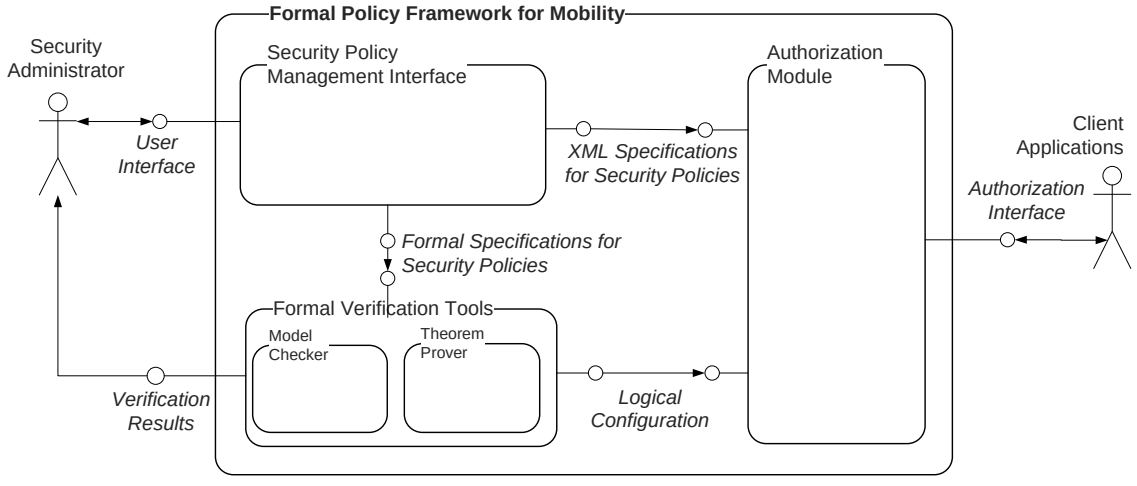

Figure 3.3. Block diagram of Formal Policy Framework for Mobility.

The Formal Policy Framework for Mobility (FPFM) provides an integrated framework for specification, enforcement and verification of security policies for multi-

domain mobile networks. FPFM is geared towards multi-domain, location and mobility issues in multi-domain mobile networks. FPFM provides formal models, data structures and algorithms for construction of a policy system for a specific communication or information system. The formal model of security policies within FPFM is called FPM-RBAC, which is presented in Chapter 4. Data structures are defined by the XML based specification called XFPM-RBAC, which is presented in Chapter 5.

The overall structure of FPFM is presented in Figure 3.3. FPFM consists of a Security Policy Management Interface, Formal Verification Tools and an Authorization Module. Prototype implementations of the Security Policy Management Interface and the Formal Verification Tools are provided within the framework, whereas the Authorization Module needs to be implemented according to the specific system that will adopt FPFM.

The Security Policy Management Interface provides a Graphical User Interface (GUI) which interfaces with a security administrator for definition of domain and inter-domain security policies. The Front-End generates formal representations of security policies as an input to the Formal Verification Tools and XML policy specifications as an input to the Authorization Module. The Security Policy Management Interface is presented in Section 3.3.

The Formal Verification Tools consist of a model checker and a theorem prover. We provide a summary of the Formal Verification Tools in Section 3.4. The model checker is used to model the location and mobility related aspects of security policies and to check satisfaction of location and mobility constraints against the current state of the network. We present algorithms and an implementation of a spatial model checker in Chapter 6. The theorem prover is used for consistency checking of policy rules. We utilize the Coq interactive theorem prover within FPFM. We present FPFM formal specifications for theorem proving in Chapter 7. The Verification Back-End generates formal proofs for the formal specifications as well as the logical configuration of security policies as an input to the Authorization Module.

An Authorization Module is a module which gives authorization decisions for access requests and needs to be implemented in devices or software services which use FPM-RBAC policy specifications. The FPFM framework contains a XML-based policy specification model which implements the formal FPM-RBAC model. An Authorization Module inputs XML definitions for security policies for giving authorization decisions. An Authorization Module may use the formal verification back-end for checking validity of constraints within security policies. As part of this thesis, no prototype implementation of an Authorization Module is provided. Algorithms for giving access decisions are provided as part of the FPM-RBAC model. System-specific implementations of the presented algorithm may be done according to a specific system on which FPFM is implemented (for example, a mobile network, a service based software system or a specific security product).

3.3. Specification of Security Policies in FPFM

The Security Policy Management Interface (SPMI) is an application used for defining home domain and inter-domain configurations and security policies. The home domain is the domain which hosts domain and inter-domain services and is managed by a security administrator. Domain configuration, inter-domain configuration, home domain security policy, inter-domain security policy and their associated location constraints are defined graphically in an user-friendly manner. The tool supports automatic derivation of formal specifications of security policies for the purpose of model checking. The output of the tool are security policies specified on the basis of XML schemas of the XFPM-RBAC security policy language.

3.3.1. Specification of Domain Security Policies

The configuration of a domain with the SPMI is shown in Figure 3.4. The configuration of a domain conveys the following information:

- The logical topology of the overall network (domains)
- The users, hosts, objects and object types (networks, applications, databases,

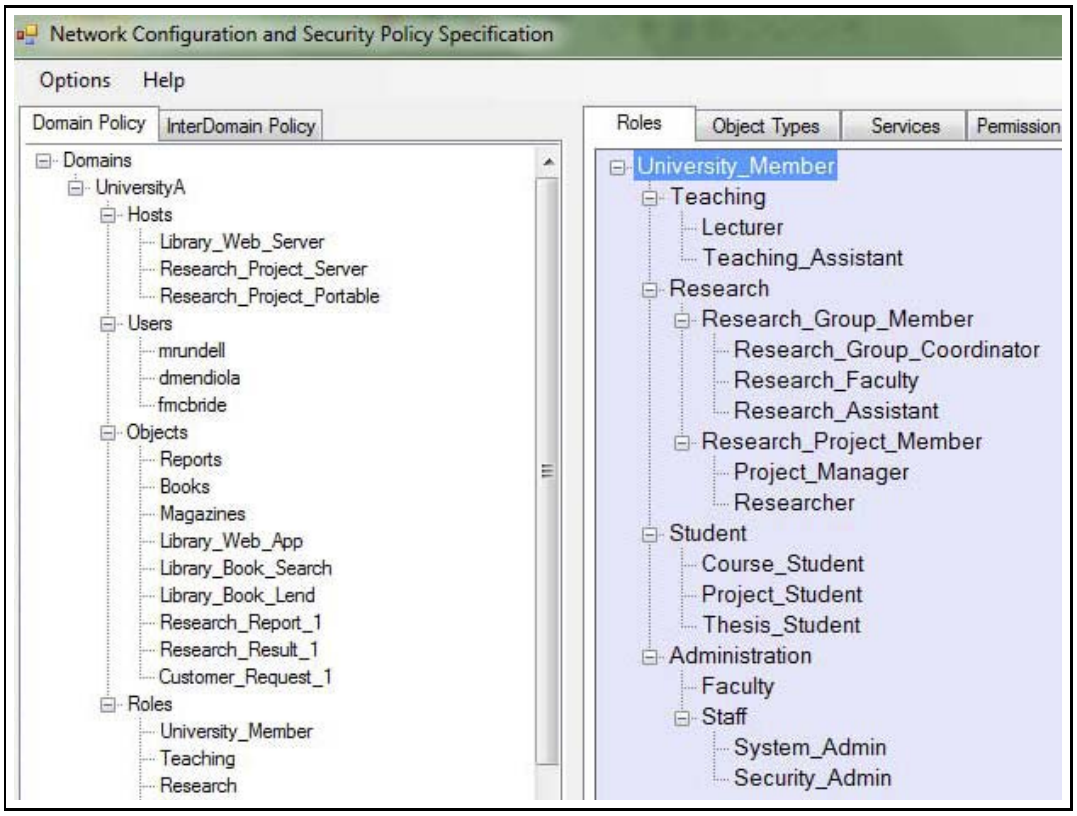

Figure 3.4. Domain configuration with the Security Policy Management Interface.

files) within the domains

- Role hierarchies and object type hierarchies

Following the definition of domain configuration, the domain security policy is defined in an iterative procedure. First, domain services are defined. Domain services are a set of authorization objects that define the set of accessible resources within the domain. The formal definition of a service is of the form $\mathcal{S}_j = \{\mathcal{D}_i, H_i^j, O_i^j, T_i^j\}$. Thus, when defining a new service, the user specifies the subset of Domains, Hosts, Objects and Object Types associated with the service. The user also selects a subset of actions which may be performed on the authorization subjects. These actions are associated with the service definition. The interface for definition of a new service is depicted in Figure 3.5.

The Service Access Matrix (SAM) defines the services that can be used by specific roles. This is a high level view of the policy. If a specific role is not allowed to use a specific service, prohibition is stated at this level without giving the details of which

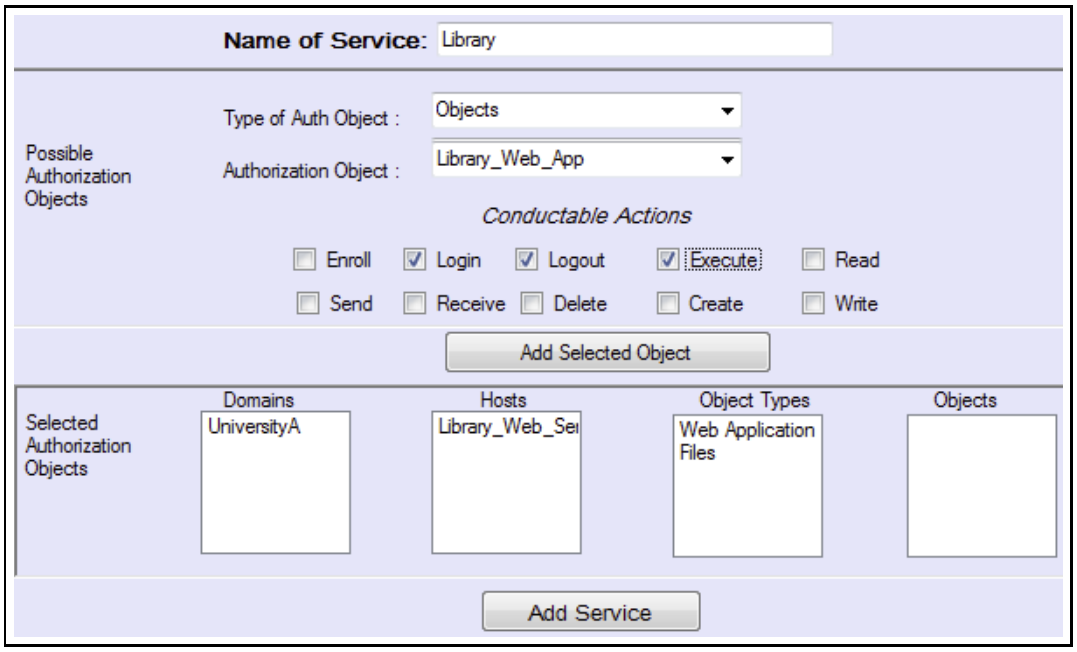

Figure 3.5. Definition of a new service.

resources are involved in accessing the service. A part of an example SAM is presented in Table 3.4.

Table 3.4. Service Access Matrix.

Role	**Services**		
	Internet	Joint Project	Library
Project Manager	Yes	Yes	Yes
Faculty	Yes	Yes	Yes
Lab Admin	Yes	Yes	No
Researcher	No	Yes	No

Permissions are specified through Permission Assignment Matrices (PAMs). A PAM is defined for each Authorization Object associated with each Service. An example PAM is shown in Table 3.5.

3.3.2. Specification of Location and Mobility Constraints

The next step in defining security policy is specification of location and mobility constraints. Location and mobility constraints for security policy rules define the spatial and temporal conditions that need to be satisfied at a given network configuration.

Table 3.5. Part of the Permission Assignment Matrix.

Role	**Actions**		
	login	logout	execute
Project Manager	Yes	Yes	Yes
Faculty	Yes	Yes	No
Lab Admin	No	No	Yes
Researcher	Yes	No	No

One can specify the location from which a permission may be invoked. Also, specific mobility constraints may be added. As an example, graduate students can be banned from accessing research project files while they visit another campus, however they may access these files from their home domain. The location constraints are specified using Ambient Logic. However, the security administrator does not need to know this formal language since formal statements are constructed from within the user interface. The specification of location constraints using the user interface is shown in Figure 3.6.

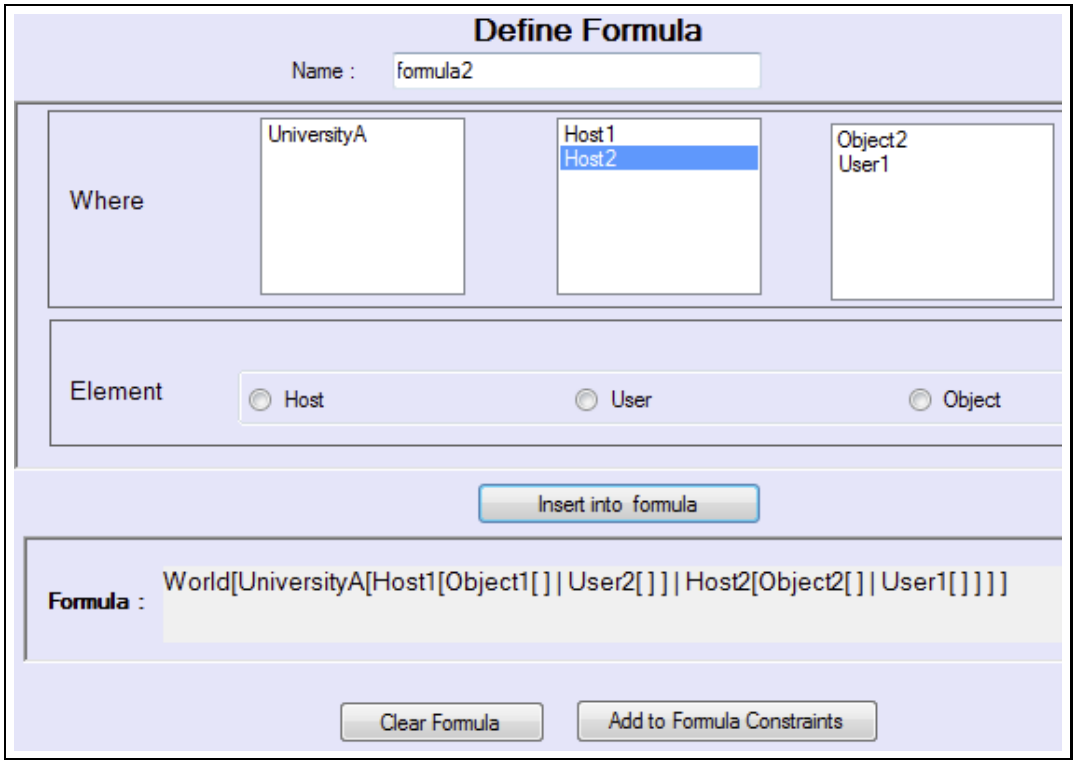

Figure 3.6. Definition of location constraints.

The Authorization Terms Matrix associates permissions with location constraints as well as generic constraints. Each row of the Authorization Terms Matrix is an

authorization term. An authorization term is of the form $at = (as, ao, sa, co, fo)$. Here, *as* is an Authorization Subject (Roles or Users), *ao* is an Authorization Object (Objects, Hosts, Domains), *sa* is a signed action, *co* is a Predicate Logic formula representing generic constraints, and *fo* is an Ambient Logic formula representing Location constraints. A part of an Authorization Terms matrix is shown in Table 3.6. In the first term, a *Student* role is given *login* access to the *Library Web Server*, with the constraints that the *Student* must be an *Enrolled Domain User* of *UniversityA* domain and the location of the *Student* must be within the *UniversityA* domain. In the second term, a *Researcher* role is given access right to write to *Research Report*, provided that the *Researcher* has already logged into the *UniversityA* domain and the *Research Report* is located within the *Research Project Server*.

Table 3.6. Part of the Authorization Terms Matrix.

A.Subject	**A.Object**	**S.Action**	**Condition**	**Formula**
Student	Lib_Web_Server	$+login$	$EnrolledDomainUser$ $(Student, UniversityA)$	$World[UniversityA[Lib\_Web\_Server[]\|$ $Student[]\|T]\|T]$
Researcher	Research_Report	$+write$	$ActiveDomainUser$ $(Researcher, UniversityA)$	$\diamond\{Research\_Project\_Server$ $[Research\_Report]\}$

3.3.3. Specification of Inter-Domain Security Policies

Inter-domain security policy consists of Inter-Domain Role Hierarchy, Role Maps, Inter-Domain Services, Inter-Domain Service Access Matrix, Inter-Domain Permission Assignment Matrix, and Inter-Domain Authorization Terms. Foreign role hierarchies are exported by domain configuration files of foreign domains. They are defined by a foreign domain administrator. Subsequently, they are imported by home domain administrator into the security policy specification interface of the home domain. This way, a domain administrator only needs to share the role hierarchy with other domain administrators and the autonomous administration is provided. The knowledge of objects and user identities are not shared among domains and accessed are provided by role mapping.

The role hierarchies of foreign and home domains are mapped to the Inter-Domain Role Hierarchy using Role Maps. The interface for this operation is shown in Figure 3.7. One or more home and foreign roles may be mapped onto one inter-domain

role using Role Maps. In the example specification presented in Figure 3.7, three roles (*Thesis_Student, Project_Student, Course_Student*) from the home domain *UniversityA* and three roles (*Grad_Student, Undergrad_Student, Student*) from the foreign domain are mapped onto the inter-domain role *Guest_Student*.

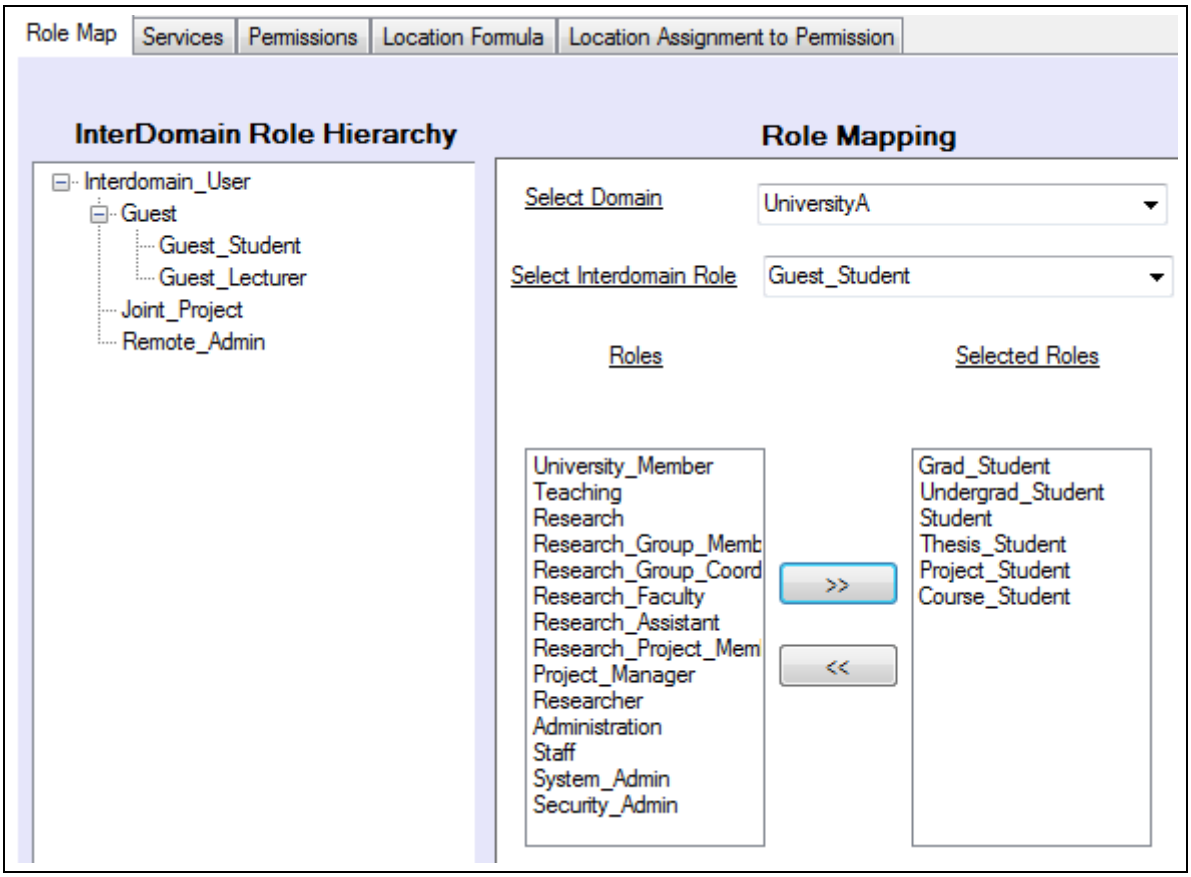

Figure 3.7. Definition of inter-domain role hierarchies and role maps.

After mapping home and foreign domain roles to inter-domain roles, inter-domain services and inter-domain permissions need to be specified. Inter-domain services define a subset of home domain authorization objects which may be used by inter-domain roles. Inter-Domain Permission Assignment Matrix maps inter-domain roles to permissions. The Inter-Domain Service Access Matrix and Inter-Domain Permission Assignment Matrix are specified in similar manner to their counterparts presented in Section 3.3.1. However, inter-domain roles are used as authorization subjects instead of domain roles.

The location and mobility constraints specification for inter-domain policies includes multiple domains and foreign users represented by inter-domain roles. Aside

from this difference, their specification is similar to Section 3.3.2. After this step, the inter-domain location and mobility constraints and generic constraints are assigned to permissions through the Inter-Domain Authorization Terms Matrix. This matrix is similar to the Authorization Terms Matrix except the replacement of domain roles by inter-domain roles.

3.4. Formal Verification of Security Policies in FPFM

Security policies in multi-domain environments are quite complex for verification with any single available formal method. Specification and verification of such a complex architecture requires formal methods that enable both static and dynamic specification. An integrated formal environment suitable for our modeling approach has been proposed. We make use of the following formal specification and verification approaches:

- Behavioral, state based specification of mobile network model
- Logic and model based based specification for security policy specification
- Model checking and automated theorem proving to specify and prove security properties for a specific security policy and network configuration

Now we will discuss alternatives for realization of such an approach and provide rationale on the method utilized in this thesis.

For specifying the static aspects of security policies, model based specification languages such as Z, B, or VDM are alternatives. Z is an established standard and has been successfully applied to security policy languages. Schemas allow formal mapping from grammar to a policy specification. Z allows representation of data structures as well as hierarchical relations which are essential elements for policy rule bases. Another alternative is to define a XML-based specification language. XML is an industry standard which is well suited for the representation of data structures and hierarchical relations. Compared to Z, B or VDM, XML does not have a formal basis. However, it is possible to define a formal language on top of XML. Because of its versatility

and wide-spread use in networks and service based applications, we opt for the use of XML as the basis of our security policy specification language. We define formal specifications as XML schemas by making use of the support for schema definitions in XML.

For specifying dynamic architectural properties such as actions in security policy including mobility, we follow an approach based on process calculus. Among many types of calculi, for concurrent computing, CCS and CSP are widely used. Another class of calculi, nominal process calculi offers strong support for security and mobility. Pi calculus, Security pi calculus and Ambient Calculus are good candidates for our modeling approach. We choose to use the Ambient Calculus because it provides natural means to model administrative domains, subjects, objects in our model. Different variations of the Ambient Calculus, like boxed ambients, secure safe ambients provide specification and verification for complex security properties of mobile systems. We use the basic Ambient Calculus for simplicity and efficiency of the model checking algorithm.

In order to be able to reason about a set of security policies within a given set of network configurations, we need to translate security policy specifications and network configurations to formal languages. There are two distinct approaches for combining formal methods. The first approach, "shallow embedding", incorporates specification in one formal language into another. There is no formal basis for the resulting language itself. Integration is based on common naming and logical relations, each language and formal method carries its own properties. The second approach is to develop an integrated formal method. This requires devising an integrated method that binds together model-based specification and process calculi. Integration is under a formal basis, the result produces a new formal specification language and method.

In this thesis we use the "shallow embedding" approach. Formal security policy specifications contain specifications in Ambient Calculus as well as Predicate Logic. Specifications in two different formal methods are verified using different formal tools. Combination of specifications is achieved at a higher level of abstraction. For formal

verification, logical expressions are put forward to enable checking of satisfiability of a security policy specification with respect to a network configuration specification. These are accomplished by parallel use of theorem proving and model checking.

The Ambient Calculus and Ambient Logic are used to model mobility aspects in security policies. A formal structure is proposed, named authorization term. Each authorization term contains formal statements specified with Ambient Logic. The satisfiability of formal statements within security policy rules with respect to network configuration is achieved with spatio-temporal model checking. Spatio-temporal model checking is based on satisfiability of Ambient Logic formulas with respect to Ambient Calculus Specifications.

Conflicts within a security policy occur when a given security policy set results in conflicting decisions for a given access requests. The Coq theorem prover is used to specify the formal policy system and the security policy for verification of security policies with respect to conflicts. For this purpose, security policies are specified in the formal language of Calculus of Inductive Constructions (CIC).

4. FPM-RBAC: A FORMAL ROLE BASED ACCESS CONTROL MODEL FOR SECURITY POLICIES IN MULTI-DOMAIN MOBILE NETWORKS

The formal model that we present in this chapter represents static and dynamic aspects of security policies in multi-domain mobile networks. The presented security policy model supports the specification of Role-Based Access Control (RBAC) policies with Role Hierarchy, Object Type Hierarchy and Role Mapping in a multi-domain setting. We define a formal model for network state related to locations and provide the mapping of actions to the formal model. Specification of location and mobility constraints as well as Separation-of-Duty (SOD) constraints is supported.

FPM-RBAC components are, a domain security policy model, an inter-domain security policy model, a location and mobility model and a separation of duty model. The core FPM-RBAC model includes constructs for definition of a domain security policy. We introduce the concepts of domain, service and Object Type Hierarchy to the RBAC model. The domain security policy model is presented in Section 4.1.

An inter-domain security policy includes security requirements for mobility, role mapping and inter-domain access rights among multiple domains. Inter-domain security policies facilitates the provision of services between domains by providing a common set of security policies for cross-domain services. Use of inter-domain policies adds a layer of abstraction to security management by enabling the security management of cross-domain services independently from domain services. The inter-domain security policy model is presented in Section 4.2.

The representation of location and mobility aspects for objects, users, host and domains is necessary in a multi-domain mobile network. Location and mobility model of FPM-RBAC is related with representation of location and mobility constraints in the security policy rules as well as the representation of actions in the security policy

rules and the state of the network with respect to locations. The location and mobility model of FPM-RBAC is presented in Section 4.3.

We present the formalization process of security policies through some examples in Section 4.4. Generic, domain and inter-domain security policy rules are presented with examples.

Separation of Duty (SOD) constraints are related to assignment and activation of roles. Static SOD enforces constraints on the assignment of Users to Roles. If a user is authorized as a member of one role, the user may be prohibited to become a member of another (conflicting) role. Dynamic SOD enforces constraints on the activation of Roles by Users. A user may be assigned to two or more conflicting roles but may activate only one of them at any specific session. In FPM-RBAC, we introduce additional classes of SOD constraints: (i) Service based SOD, (ii) Inter-Domain SOD and (iii) Location and Mobility Based SOD. SOD constraints in FPM-RBAC are presented in Section 4.5.

In Section 4.6, we present an algorithm for giving the permission or denial decisions for requested actions against FPM-RBAC security policy specifications. In Section 4.7 we compare FPM-RBAC to the original RBAC model and to extensions of the RBAC model which cover spatial and temporal aspects.

4.1. Domain Security Policy Model

A *domain* is defined by the data sets and relations associated with an administrative boundary. A security policy is defined by a set of *authorization terms*. Authorization terms include *authorization subjects*, *authorization objects*, *actions* and *constraints*. *Services* provided by the domain to its internal and external users are also included in the authorization model. Services are associated with a subset of authorization objects. The set of *enabled roles* for a service is defined by the *service access* relation. For each service, a *permission assignment* relation is specified for association of enabled roles for the service with permissions. Constraints are specified by formal

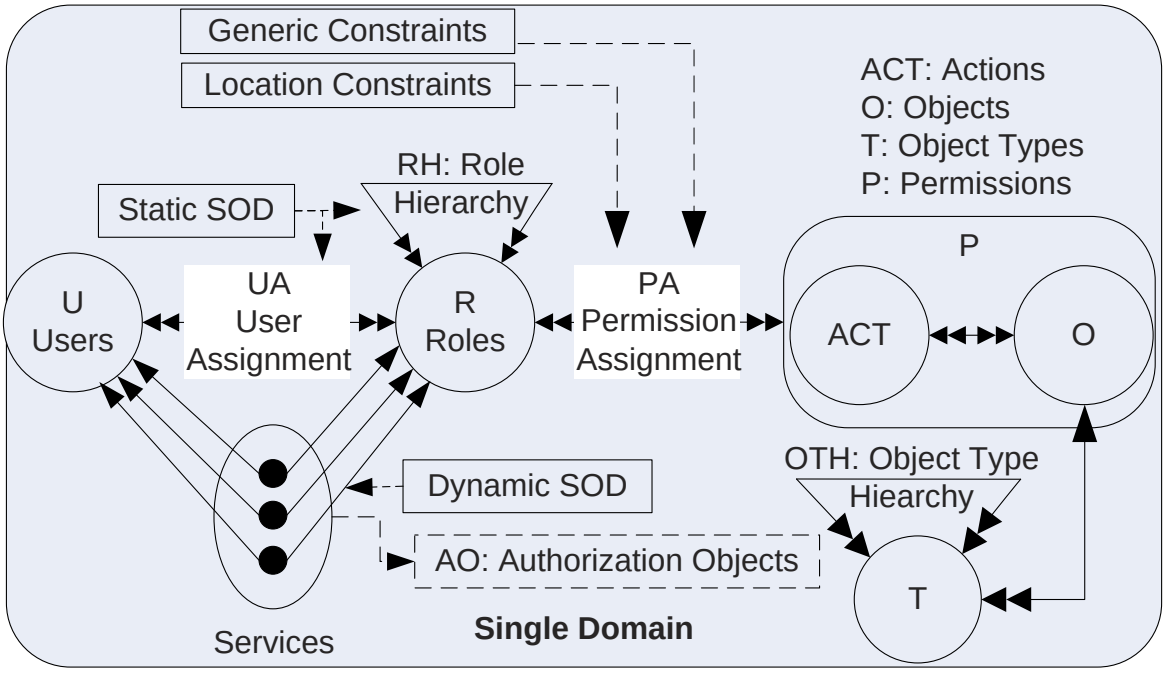

Figure 4.1. FPM-RBAC domain security policy model.

languages. Predicate calculus is used for the specification of generic constraints named as *conditions* and Ambient Logic is used for the specification of location and mobility constraints named as location formula, or *formula* in short. The single domain security policy model of FPM-RBAC is depicted in Figure 4.1. In the following sections we present data sets, domains, services, actions, constraints, relations, hierarchies of the domain security policy model.

4.1.1. Data sets

The data sets in the FPM-RBAC access control model is specified using First Order Set Theory. An Authorization Subject is an active entity that may conduct an Action on an Authorization Object. Since the RBAC model is adopted, roles are authorization subjects, however specification of users in this context are also supported. An authorization object is an entity upon which an action is conducted. An authorization object may be a domain, host, object or object type. Where the constants $\kappa, \upsilon, \rho, \tau, \zeta, \sigma$ respectively denote the number of hosts, users, roles, objects, object types and services, the data sets are defined in Table 4.1.

Table 4.1. The data sets in FPM-RBAC.

Elements of data set	**Name of data set**
$H = \{h_1, h_2, ...h_\kappa\}$	Hosts
$U = \{u_1, u_2, ...u_\upsilon\}$	Users
$R = \{r_1, r_2, ...r_\rho\}$	Roles
$O = \{o_1, o_2, ..., o_\tau\}$	Objects
$T = \{t_1, t_2, ...t_\zeta\}$	Object types
$AS = U \cup R$	Authorization Subjects
$AO = O \cup T \cup H \cup \Gamma$	Authorization Objects

4.1.2. Domains

A domain is defined by a set of hosts, users, objects, roles and object types associated with that particular domain. Each domain is associated with a domain administrator. A Domain $\mathcal{D}$ is defined as $\mathcal{D} = \{H, U, O, R, T\}$.

When there are multiple domains in the network, δ represents the number of domains and Γ defines the set of all domains in the network. In this case, the set of hosts, users, roles, objects, object types, authorization subjects and authorization objects associated with domain $\mathcal{D}_i$ are represented respectively with $H_i, U_i, R_i, O_i, T_i, AS_i, AO_i$. The set of all domains is defined as $\Gamma = \{\mathcal{D}_1, \mathcal{D}_2, ...\mathcal{D}_\delta\}$

4.1.3. Services

The W3C Web Service Glossary defines services as follows [63]: "A service is an abstract resource that represents a capability of performing tasks that form a coherent functionality from the point of view of providers entities and requesters entities". The FPM-RBAC model includes a specific construct for services. The Service construct binds the dynamic user-role association to a subset of resources. In the context of security policies for multi-domain mobile networks, we treat services as a set of authorization objects that define the set of accessible resources for provision of a capability.

A service in the FPM-RBAC model is associated with a subset of authorization objects, namely, Domains, Hosts, Objects and Object Types. The set of Services $\mathcal{V}$ defines the services within a single domain. If there are multiple domains, the services within a domain $\mathcal{D}_i$ is represented as $\mathcal{V}_i$. The subset of domains and hosts associated with a service represents logical locations that a role may be enabled. The subset of objects and object types associated with a service are included in the Permission Assignment Matrix related with the service. A Service $\mathcal{S}_j$ within a domain $\mathcal{D}_i$ and $\mathcal{V}_i$, the set of Domain Services within a domain $\mathcal{D}_i$ are defined as follows.

Definition 4.1. *A* service *is* $\mathcal{S}_j = \{\mathcal{D}_i, H_i^j, O_i^j, T_i^j\}$ *where* $\mathcal{D}_i$ *is the domain that* $\mathcal{S}_j$ *is associated,* $H_i^j = \{h_k, \ldots, h_l\}$ *are hosts associated with service* $\mathcal{S}_j$*,* $H_i^j \subset H_i$*,* $O_i^j = \{o_m, \ldots, o_n\}$ *are objects associated with service* $\mathcal{S}_j$*,* $O_i^j \subset O_i$*,* $T_i^j = \{t_x, \ldots, t_y\}$ *are object types associated with service* $\mathcal{S}_j$*,* $T_i^j \subset T_i$*. The set of domain services is* $\mathcal{V}_i = \{\mathcal{S}_j, \ldots, \mathcal{S}_k\}$*, where* $1 \leq i \leq \delta, j, k : 1..\sigma$*.*

Sessions in the NIST RBAC model is replaced by the *Service Sessions* in FPM-RBAC. The service sessions in the FPM-RBAC model are defined when a user is registered to a service, and activated when the user requests to use the service.

4.1.4. Actions

The set of actions defines the operations by subjects on objects. For this study we assume a fixed set of actions $A = \{Enroll, Login, Logout, Execute, Read, Write, Send, Receive, Delete, Create, Manage\}$. The action *Manage* is an administrative action relating to the policy system. The set of permissions $P = ACT \times AO$ defines all possible actions on authorization objects for a given system.

The actions may also be signed. Positive authorizations are considered default and positive sign may be omitted within specifications. The following constructs are used to specify signed actions:

- Signs: $N = \{+, -\}$ represents permission or denial.

- Signed Actions: $ACT = N \times A$ represents permission or denial of an action. (+,read) denotes that read action is permitted.

4.1.5. Constraints

A *constraint* determines the applicability of a security policy rule. The set of constraints is of the form $C = \{(co, fo) : co \in PL, fo \in AL\}$ where PL denotes the language of predicate logic and AL denotes the language of Ambient Logic. There are two types of constraints. Conditions (co) define the generic constraints for a rule to be applicable. Location formula (fo) defines the location and mobility constraints of the mobile network that must be satisfied by a security policy rule. Location and mobility constraints are discussed in Section 4.3.

Generic constraints are logical pre-requisites for a rule to be applicable and are defined using predicate logic. Generic constraints are specified by a predicate logic formula that defines the constraints and relationships between entities specified in the policy rules. In cases where roles are mentioned in a rule, generic constraints may apply to user-role assignment.

We define the following pre-defined predicates for definition of generic constraints. Additional predicates may be defined by the domain administrator.

(i) EnrolledDomainUser $(\mathcal{D}_i, u_j)$, where $\mathcal{D}_i \in \Gamma, u_j \in U$, specifies whether a user u_j has been enrolled (registered) to domain $\mathcal{D}_i$. Abbreviated as $EDR(\mathcal{D}_i, u_j)$.

(ii) EnrolledDomainHost $(\mathcal{D}_i, h_k)$, where $\mathcal{D}_i \in \Gamma, h_k \in H$, specifies whether a host h_k has been enrolled (registered) to domain $\mathcal{D}_i$. Abbreviated as $EDH(\mathcal{D}_i, h_k)$.

(iii) ActiveDomainUser $(\mathcal{D}_i, u_j)$, where $\mathcal{D}_i \in \Gamma, u_j \in U$, specifies whether user u_j has been logged into domain $\mathcal{D}_i$. Abbreviated as $ADU(\mathcal{D}_i, u_j)$.

(iv) RoleAssigned (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. Abbreviated as $RSG(u_j, r_l)$.

(v) RoleAssumed (u_j, r_l), where $u_j \in U, r_l \in R$, specifies whether user u_j has actively assumed role r_l. Abbreviated as $RAS(u_j, r_l)$.

(vi) RoleEnabled $(r_l, \mathcal{S}_m)$, where $\mathcal{S}_m \in V, r_l \in R$, specifies whether role r_l is enabled for service $\mathcal{S}_m$. Abbreviated as $REN(r_l, \mathcal{S}_m)$.

(vii) DescendantRole (r_l, r_k), where $r_l, r_k \in R$ specifies whether a given role is r_l descendant (or specialization) of another given role r_k. Abbreviated as $DR(r_l, r_k)$

(viii) ObjectIsType (o_n, t_m), specifies whether an object $o_n \in O$ identified by its object name is of a given object type $t_m \in T$. Abbreviated as $OIT(o_n, t_m)$

(ix) Administrator $(u_j, \mathcal{D}_i)$, where $u_j \in U, \mathcal{D}_i \in \Gamma$ specifies whether user u_j has administrative rights over domain $\mathcal{D}_i$. Abbreviated as $ADM(j, i)$.

4.1.6. Relations and System Functions

Relations in the FPM-RBAC model are specified in Table 4.2. The HD relation maps hosts to domains, whereas UD relation maps users to their home domains. The UA relation specifies the assignment of users to roles. The PA relation specifies the assignment of roles to permissions. Finally, the SA relation specifies the set of enabled roles for services.

Table 4.2. Relations in the FPM-RBAC model.

Relation Name	**Relation**	**Specification**	**Meaning**
HD	$H \times \Gamma$	$HD(h_k, \mathcal{D}_i)$	h_k is enrolled to Domain $\mathcal{D}_i$.
UD	$U \times \Gamma$	$UD(u_j, \mathcal{D}_i)$	u_r is enrolled to Domain $\mathcal{D}_i$.
OD	$O \times \Gamma$	$OD(o_n, \mathcal{D}_i)$	o_n is within Domain $\mathcal{D}_i$.
UA	$U \times R$	$UA(u_j, r_l)$	Role r_l assigned to the user u_j.
PA	$R \times ACT \times AO$	$PA(r_l, act, ao)$	r_l has permission (act, ao).
SA	$R \times \mathcal{V}$	$SA(r_l, \mathcal{S}_m)$	r_l is enabled for service $\mathcal{S}_m$.

The permission assignment and service assignment relations are specified in terms of matrices. The dynamic binding of users to roles and permissions is achieved through services. Permissions are assigned to roles based on permission assignment matrices defined for each service and authorization object type.

Service Access Matrix binds roles to services. When a user logs into a service,

the roles associated with that service are enabled. Then the role may act only on the subset of domains, hosts, objects and object types referred by the service. The SAM matrix denotes whether role r_l is allowed to use Service $\mathcal{S}_j$.

Definition 4.2. *The Service Access Matrix is* $SAM : R \times \mathcal{V}$ *where* $SAM[i,j] = True \mid False$, $1 \leq i \leq \rho, 1 \leq j \leq \sigma$. $SAM[i,j] = True \rightarrow SA(r_i, \mathcal{S}_j)$.

A Permission Assignment Matrix is defined for each service, which also includes Location and Mobility constraints. This design decision simplifies the specification of permissions for a network with multiple domains and different types of services. Permission Assignment Matrix binds roles to permissions. Permissions are tuples of the form (act, ao) where $act \in ACT, ao \in AO$. For each service there is an Permission Assignment Matrix that includes permissions for that service. $PAM[i,j,k]$ determines whether Role r_i is allowed to conduct Action act_j on Authorization Object ao_k.

Definition 4.3. *The Permission Assignment Matrix is* $PAM : R \times ACT \times AO$ *where* $PAM[i,j,k] = True \mid False$, $1 \leq i \leq \rho, 1 \leq j \leq |ACT|, 1 \leq k \leq |AO|$. $PAM[i,j,k] = True \rightarrow PA(r_i, act_j, ao_k)$.

The system functions in the FPM-RBAC model are defined in line with the standard RBAC model [54, 55]. Additional functions are provided because of introduction of services and object type hierarchies. The concept of sessions is enriched with introduction of services and service sessions. The functions in FPM-RBAC are presented in Table 4.3.

4.1.7. Role and Object Hierarchies

Hierarchies in the FPM-RBAC model consist of Role Hierarchy (RH) and Object Type Hierarchy (OTH). The precedence relationship of roles are defined in the Role Hierarchy (RH) relation. The mapping of objects to object types are defined in the Object Type Hierarchy (OTH) relation. The Object Type Hierarchy is compatible with the Flexible Authorization Framework (FAF) [23] and the Role Hierarchy is compatible with the Role-Based Access Control [54, 55] model. When there are multiple domains,

Table 4.3. System functions in the FPM-RBAC model.

Function Name	**Specification**	**Meaning**
services	$U \rightarrow 2^V$	Gives the services accessible by a user.
service_users	$V \rightarrow 2^U$	Maps each service to its users.
enabled_roles	$V \rightarrow 2^R : \{r \in R \mid (v,r) \in SA\}$	Maps each service to the set of enabled roles.
enabled_roles*	$V \rightarrow 2^R : \{r \in R \mid r' \succeq r, (v,r') \in SA\}$	Maps each service to the set of enabled roles in presence of a role hierarchy.
assigned_users	$R \rightarrow 2^U : \{u \in U \mid (u,r) \in UA\}$	Maps a role to set of users.
authorized_users	$R \rightarrow 2^U : \{u \in U \mid r' \succeq r, (u,r') \in UA\}$	Maps a role to set of users in presence of role hierarchy.
object_type	$O \rightarrow T$	Gives the type of an object.
roles	$U \cup P \cup V \rightarrow 2^R$	Maps users, permissions and services to roles.
roles*	$U \cup P \cup V \rightarrow 2^R$	Maps users, permissions and services to roles in presence of role hierarchy.
permissions	$R \rightarrow 2^P$	Maps roles to permissions.
permissions*	$R \rightarrow 2^P$	Maps roles to permissions in presence of role hierarchy.

the role and object type hierarchies for domain $\mathcal{D}_i$ are represented as RH_i and OTH_i respectively.

Definition 4.4. *Hierarchy. A* hierarchy *is a triple (A, B, $\preceq$) where*

(i) A and B are disjoint sets;

(ii) $\preceq$ is a partial order on A$\cup$B s.t. every element a$\in$A is said to be minimal in

$A \cup B$. Minimal elements have no elements below themselves in the hierarchy, i.e. $a \in A$ and $\forall b \in A \cup B$, $b \preceq a \Rightarrow b = a$.

For the OTH relation, $a \preceq b$ means that object a is of object type b. For the RH, $a \preceq b$ means that role a is a specialization of role b.

Definition 4.5. Object Type Hierarchy. *$A=O$, $B=T$, $a \preceq b \rightarrow (a, b) \in OTH$ where $a \in O, b \in T$.*

Definition 4.6. Role Hierarchy. *$A=R$, $B=R$, $a \preceq b \rightarrow (a, b) \in RH$ where $a \in R, b \in R$.*

The types for Services may be defined as sub-types for Application object type in the Object Type hierarchy. The use of services together with Object-Type Hierarchies enable the use of FPM-RBAC model in specification of security policies for software, network and security services.

4.1.8. Authorization Terms

An authorization term *at* is the basic formal construct used to specify security policy rules. AT is the set of all authorization terms and AT_i is the set of authorization terms for Domain $\mathcal{D}_i$. The Permission Assignment relations together with Generic Constraints and Location and Mobility Constraints are formalized by authorization terms. Set of authorization terms AT is defined as: $AT \subset PA \times C$ where $at \in AT$, $AT = \{(as, ao, act, co, fo) : as \in AS, ao \in AO, act \in ACT, (co, fo) \in C\}$.

4.1.9. Domain Security Policy

The set of defined services, authorization terms and hierarchies that constitutes the security policy for a domain $\mathcal{D}_i$ is named $\mathcal{P}_i$. The data sets that are referred by the domain security policy are included in the specification of $\mathcal{D}_i$. The domain security policy for a domain $\mathcal{D}_i$ is defined as $\mathcal{P}_i = \{\mathcal{V}_i, AT_i, RH_i, OTH_i\}$, $1 \leq i \leq \delta$.

Since the FPM-RBAC model is multi-domain, there may be multiple domain security policy definitions. Each domain security policy defines its access rules for local users to access local resources within the domain. The access rules for users to access inter-domain resources are specified by an inter-domain security policy. As a result of this administration method, the knowledge of local users and resources remains within the administrative boundary of a domain. The set of Domain Security Policies in a multi-domain environment is defined by the set Ω. The set of domain security policies is defined as $\Omega = \{\mathcal{P}_1, \ldots, \mathcal{P}_\delta\}$.

4.2. Inter-Domain Security Policy Model

The constructs specified in the Domain Security Policy Model presented in Section 4.1 are also valid in the Inter-Domain Security Policy Model (hereafter referred as inter-domain model). The inter-domain model adds constructs for *inter-domain services*, *inter-domain roles*, *inter-domain role hierarchy*, *role maps* and the *inter-domain service access* and *inter-domain permission assignment* relations. Inter-domain security policy is defined over inter-domain authorization terms which are specified on the basis of these constructs.

The inter-domain model introduces the concepts of *Home Domain*, *Foreign Domain* and *Inter-Domain*. Home Domain is the boundary of security administration for a domain which provides services to other domains. Foreign domains are domains whose users access inter-domain services through mapped roles. Inter-domain policy includes rules for foreign users to access home domain objects through mapped roles, as well as the mapping of home and foreign domain roles to inter-domain roles.

As far as domain security administration is concerned, home domain administrator is only informed about the inter-domain role hierarchy. The home domain administrator defines rules for the foreign users to access home domain objects. In this manner, the principle of autonomous administration is preserved without the need of global distribution of knowledge of objects and users.

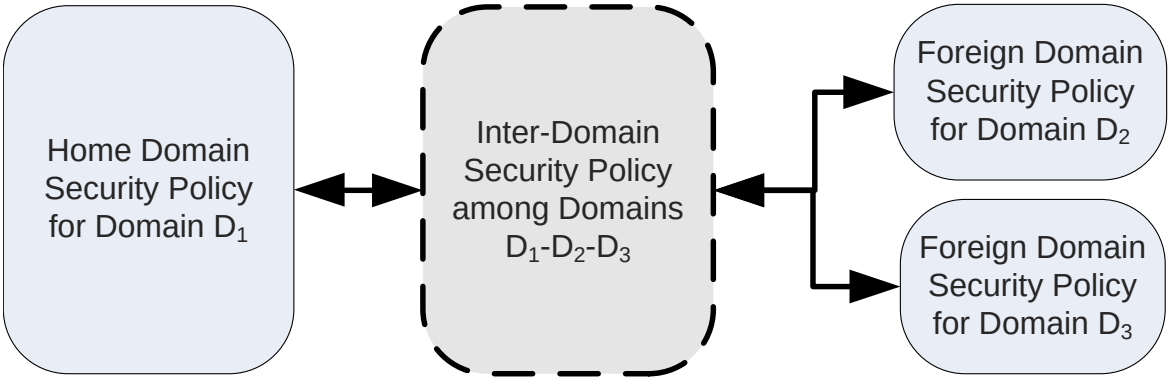

Figure 4.2. Access control policies in multi-domain environment.

The overall view for access control policies in a multi-domain environment is depicted in Figure 4.2. The inter-domain access control model of FPM-RBAC is detailed in Figure 4.3. Here, we introduce the concept of Inter-Domain Roles and Inter-Domain Role Hierarchies. Roles are mapped to Inter-Domain Roles with a role mapping function. An inter-domain service is accessible by home or foreign users through inter-domain roles and includes objects from home domain. The access rules for inter-domain services by foreign roles is achieved through the *Inter-Domain Permission Assignment* relation.

4.2.1. Inter-Domain Services

Unlike domain services, which are associated with a single domain, *inter-domain services* are services which are associated with a set of domains Γ. One of the elements of Γ is named as *home domain* and the others are named as *foreign domains*. Inter-domain services contain objects from the home domain. They are accessed by inter-domain roles. An inter-domain service $\hat{\mathcal{S}}$ is defined as follows.

Definition 4.7. Inter-Domain Service $\hat{\mathcal{S}}$ *is* $\hat{\mathcal{S}}_j = \{\Gamma, H^j, O_a^j, T_a^j\}$
where $\Gamma = \{\mathcal{D}_a, \mathcal{D}_b, \mathcal{D}_c, ...\}$ *is the set of domains in the multi-domain environment,*
D_a *is the Home Domain,* $D_b, D_c \ldots$ *are foreign domains,*
$H^j = \{h_k, \ldots, h_l\}$ *are hosts associated with* $\hat{\mathcal{S}}_j$, $H^j \subset H_a \cup H_b \cup H_c ...$,
$O_a^j = \{o_m, \ldots, o_n\}$ *are objects associated with* $\hat{\mathcal{S}}_j$, $O_a^j \subset O_a$,
$T_a^j = \{t_x, \ldots, t_y\}$ *are object types associated with* $\hat{\mathcal{S}}_j$, $T_a^j \subset T_a$

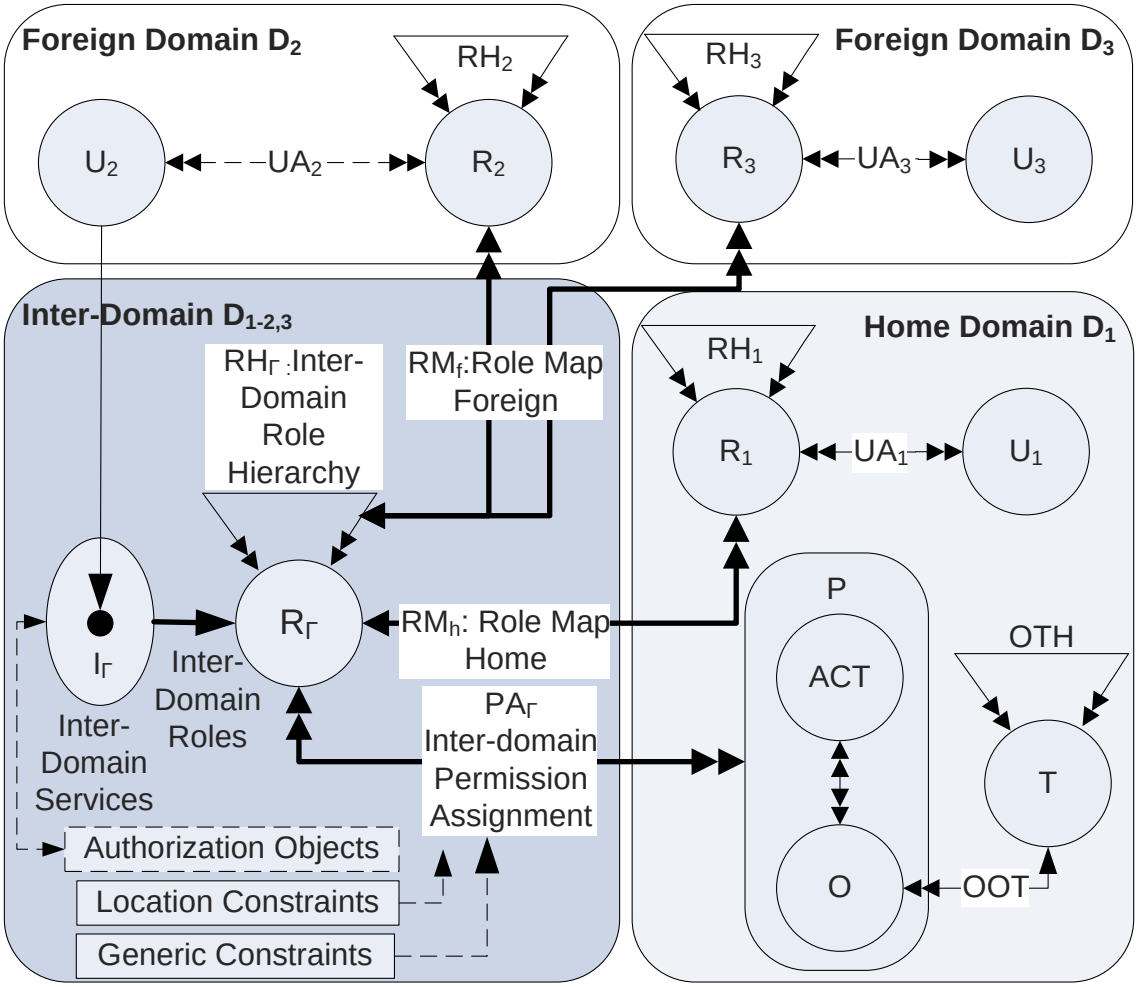

Figure 4.3. FPM-RBAC inter-domain security policy model.

The set of inter-domain services $\mathcal{I}_\Gamma$ among a set of domains Γ, where D_a is the Home Domain and $D_b, D_c \ldots$ are foreign domains, is specified as $\mathcal{I}_\Gamma = \{\hat{\mathcal{S}}_j : 1 \leq j \leq \gamma\}$ where γ is the number of inter-domain services.

4.2.2. Inter-Domain Roles

An *inter-domain role* is a role which does not apply to a particular domain and is used for mapping of roles between domains. A direct map between roles of multiple domains is avoided to provide more flexibility of administration and enforcement. Inter-domain services can be directly specified according to the needs of information sharing between organizations and enforced with respect to inter-domain roles. The set of Inter-Domain Roles R_Γ is defined as $R_\Gamma = \{r_\Gamma^1, \ldots, r_\Gamma^\varrho\}$ where ϱ is the number of inter-domain roles. The definition of inter-domain role hierarchy RH_Γ in Definition 4.8 is similar to RH relation.

Definition 4.8. Inter-Domain Role Hierarchy RH_Γ *is defined as follows:*
$RH_\Gamma \triangleq a \preceq b \rightarrow (a, b) \in RH_\Gamma,\ a, b \in R_\Gamma$.

4.2.3. Role Map

A *role map* RM is a many-to-one relation from a set of roles R to inter-domain roles R_Γ. The choice of many-to-one relations for defining role maps eliminates conflicts in role mapping. A home or foreign role may not map to multiple inter-domain roles associated with a possibly conflicting set of permissions. Furthermore a direct map between roles of multiple domains is avoided by introduction of home and foreign role maps. This decision facilitates independent administration of inter-domain roles and inter-domain role hierarchies. The role map from home roles to inter-domain roles is represented as RM_h and the role map from foreign roles to inter-domain roles is represented as RM_f.

Definition 4.9. Home and Foreign Role Maps, RM_h *and* RM_f, *are defined as follows:*
$RM_h = \{(r_i, r_\Gamma^j) : r_i \in R_a, r_\Gamma^j \in R_\Gamma\}.\ \exists(r_x, r_y) \in RM_h \wedge \exists(r_x, r_z) \in RM_h \rightarrow y = z$
$RM_f = \{(r_k, r_\Gamma^j) : r_k \in R_b, r_\Gamma^j \in R_\Gamma\}.\ \exists(r_x, r_y) \in RM_f \wedge \exists(r_x, r_z) \in RM_f \rightarrow y = z,$

where D_a is the home domain and D_b is a foreign domain.

Consider the example presented in Figure 4.4. In this example the role hierarchies are assumed to be tree structures for the sake of simplicity.

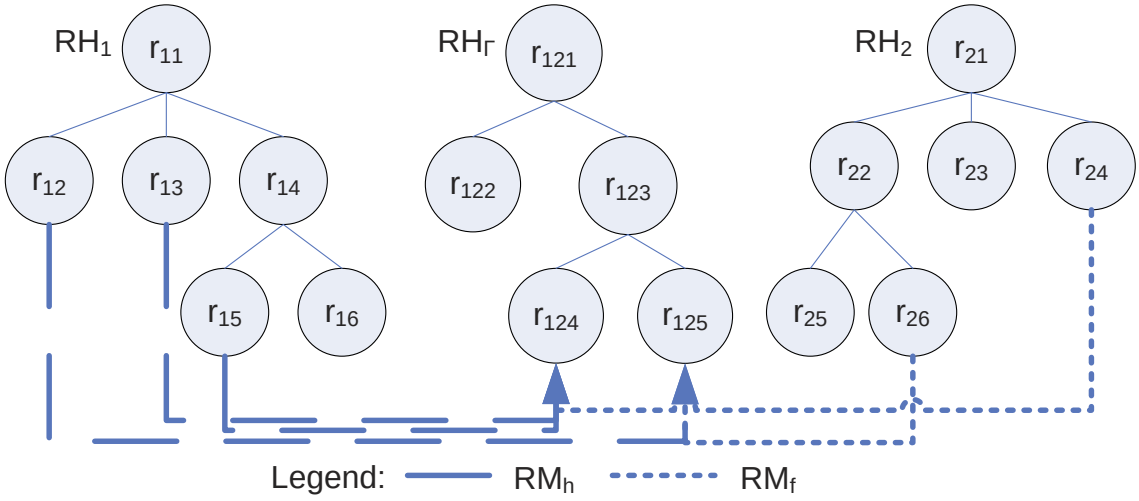

Figure 4.4. Example for home and foreign role maps among two domains.

In this example, we assume the following:

$$
\begin{aligned}
R_1 &= \{r_{11}, r_{12}, r_{13}, r_{14}, r_{15}, r_{16}\},\\
R_2 &= \{r_{21}, r_{22}, r_{23}, r_{24}, r_{25}, r_{26}\},\\
R_\Gamma &= \{r_{121}, r_{122}, r_{123}, r_{124}, r_{125}\},\\
RM_h &= \{(r_{12}, r_{125}), (r_{13}, r_{124}), (r_{15}, r_{124})\},\\
RM_f &= \{(r_{24}, r_{124}), (r_{26}, r_{125})\}.
\end{aligned}
$$

4.2.4. Inter-Domain Relations and Authorization Terms

The relation PA_Γ is the inter-domain permission assignment relation, used for the assignment of inter-domain roles to permissions. SA_Γ is the inter-domain service assignment relation used for assignment of inter-domain roles to inter-domain services.

AT_Γ is the set of inter-domain authorization terms specified as inter-domain permission assignments associated with constraints. C_Γ is the set of inter-domain constraints. Inter-domain permission assignment, inter-domain service access relation and inter-domain authorization terms are respectively specified as follows: $PA_\Gamma \subset R_\Gamma \times ACT \times AO$, $SA_\Gamma \subset R_\Gamma \times \mathcal{I}_\Gamma$, $AT_\Gamma \subset PA_\Gamma \times C_\Gamma$, $AT_\Gamma = \{(as, ao, sa, co, fo) : as \in R_\Gamma, ao \in AO, sa \in ACT, (co, fo) \in C_\Gamma\}$.

4.2.5. Inter-Domain Security Policy

An inter-domain security policy $\mathcal{W}_\Gamma$ is defined among a set of domains Γ, where D_a is the Home Domain and $D_b, D_c \ldots$ are foreign domains. $\mathcal{W}_\Gamma$ includes inter-domain services $\mathcal{I}_\Gamma$, inter-domain role hierarchy RH_Γ, home domain to inter-domain role-mapping RM_h, foreign domain to inter-domain role mapping RM_f and inter-domain authorization terms AT_Γ. The formal representation of Inter-Domain Security Policy is $\mathcal{W}_\Gamma = \{\mathcal{I}_\Gamma, RH_\Gamma, AT_\Gamma, RM_h, RM_f\}$.

4.3. Location and Mobility Model

Location and mobility model of FPM-RBAC is related with representation of location and mobility constraints in the security policy rules as well as the representation of actions in the security policy rules and the location configuration of the system. 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. These aspects are modeled in the security policy rule as *location and mobility constraints*. We utilize Ambient Logic for specification of location and mobility constraints. Ambient Calculus is used for representation of actions in the security policy rules and the location configuration of the system. The satisfaction of location and mobility constraints within the current state of the system is achieved through spatio-temporal model checking. The satisfiability relation of Ambient Logic formulas against Ambient Calculus specifications is the basis for spatio-temporal model checking [41]. We provide a very brief overview of Ambient Calculus and Ambient Logic in the following section.

Table 4.4. Fragment of Ambient Calculus used in this Thesis.

Sym.	**Syntax**	**Meaning**	**Sym.**	**Syntax**	**Meaning**
P, Q	$::=$	processes	M	$::=$	capabilities
	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
	$\langle M \rangle$	asynchronous output		ϵ	null
				$M.M$	path

4.3.1. Ambient Calculus and Ambient Logic

The proposed methodology uses Ambient Calculus for specifying multi-domain mobile network location configurations. Fragment of Ambient Calculus used in this study is presented in Table 4.4. The semantics of Ambient Calculus is based on structural congruence relation.

Like all process algebras, Ambient Calculus relies on the notion of process. Properties of Ambient Calculus processes can be analyzed in two ways, spatial properties and temporal properties. The notion of Ambient is the basic element of the spatial properties of processes. Ambients are bounded places identified by a name where processes reside inside or outside. Ambients can be nested in other ambients. This provides an hierarchical organization of locations.

Process algebras have constructs which represent the change of processes over time, called temporal constructs. Because the main aspect of Ambient Calculus is modeling mobility of the systems, it provides temporal constructs related to mobility in addition to communication primitives. There are three main temporal constructs in Ambient Calculus for modeling entrance, exit and dissolution which are expressed as *in* n, *out* n, *open* n respectively. Capabilities can be ordered as a sequence, called a *path*, to represent sequential execution as well as parallel execution represented by

the symbol (|). A specification in the form of $a[in\ n.out\ z.0]$ represents an ambient a which will enter ambient n, exit ambient z and stop. While the ability to change ambient hierarchy represents mobility, this can be used to represent any kind of computation. Communication primitives enable processes within the same ambient to exchange messages.

Names are not only used for identifying ambients but also used as access keys for ambients; capabilities can effect or use ambients with name known by them. Two distinct ambients can have the same name. But in the fragment of Ambient Calculus used in this paper names are restricted to be unique. They become an identifier in 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 parallel composition operator. It is a commutative and associative operator. In Ambient Calculus communication constructs are asynchronous and local to an ambient. Ambient calculus does not support channel names for communication. While communication is used to exchange both names and capabilities in the full fragment, communication of capabilities is excluded in this paper.

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 variable in the scope of input capability is replaced with the incoming name. The semantics of Ambient Calculus is based on structural congruence relation. Structural congruence identifies processes up to 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.

Modal logic is used for expressing properties of models which cannot be expressed

Table 4.5. Fragment of Ambient Logic used in this Thesis.

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

by the constructs of calculi. Ambient Logic [4, 32] is very expressive modal logic for expressing spatial and temporal properties of Ambient Calculus. All spatial and temporal constructs of Ambient Calculus are reflected in Ambient Logic. The main differences of Ambient Logic from latter logics are introduction of more expressive space modalities and simpler temporal connectives [64, 65].

Ambient Logic has temporal and spatial modalities in addition to propositional logic elements. Semantics of the connectives of the Ambient Logic are defined through satisfaction relations. The definition of satisfaction is based on the structural congruence relation. The satisfaction relation is denoted by $\models$ symbol. $P \models \mathscr{A}$ denotes that process P satisfies formula $\mathscr{A}$.

The Ambient Logic provides two main spatial and temporal constructs. The *Somewhere* connective, $\diamond$, is used for specifying nesting properties of processes. The formula $\diamond\mathscr{A}$ is satisfied by processes which satisfies A in some inner location. The *Sometime* connective, $\Diamond$, is used for specifying temporal behavior of the processes on the basis of reduction relations ($\rightarrow$). Fragment of Ambient Logic used in this paper is shown in Table 4.5.

4.3.2. Representation of Location and Mobility in FPM-RBAC

The mobility model for a multiple domain mobile network consists of four basic elements: administrative domains, hosts, users and objects. We assume that all the elements reside in a system element called the World. In this model the locations and mobility of hosts, users and objects are formalized as Ambient Calculus processes and they have the following mobility capabilities:

- Hosts: Moving into a domain represents connecting a host to a domain. Moving out represents disconnecting.
- Users: If users move into a host, this represents logging into a host. If a user moves out of a host, the user is logged out. If the user moves into/out of a domain, this represents enrolment, logging into a domain or movement between domains depending on the context.
- Objects: Every object must reside in a host when not on the move. The movement of objects from one host to another represents communication.

In the formal specification, domains, hosts, users and objects are modeled as Ambients. The actions are modeled as Ambient Calculus capabilities. The *in* action makes an ambient enter into another ambient. The result is that the moving ambient becomes a child node for its sibling node. For example, the path $in\ Host.File[]|Host[]$ results in the location configuration $Host[File[]]$. The *out* action causes an ambient to exit its parent ambient. The result is that an ambient moves up to the same level with its parent in the location hierarchy. For example, $Host[File[outHost.0]]$ results in $Host[]|File[]$. The other two actions, *mv_in* and *mv_out* are similar to in and out actions, except that they move other ambients instead of moving the ambient they are contained. The *acid* action dissolves the boundary of the ambient in which the action is contained. This represents deletion of an object in the network. For example, $acid\ File.File[]$ deletes the file from the formal specification. The *open* action dissolves the boundary of another ambient. For example, $User[open\ Message.Message[M]]$ results in the formal specification $User[M]$, which represents an user opening a message. The open capability is only used for messages. The reason for this restriction is that

"opening" other system elements such as hosts would violate the integrity of the model.

The translation of actions in the security policy rules to Ambient Calculus is based on a formal mapping. The translations are encoded as a template and are based on inference of specific subject and object names from the high-level specifications of security policy. Here we present template encodings for translation of actions to Ambient Calculus. The encoding is for a single domain. The presented encoding is not exhaustive since alternative encodings for the same actions in different contexts are possible.

Definition 4.10. *Ambient Calculus encoding for actions in security policy rules are specified as follows. In this encoding,* $\alpha_{as,ao}$ *represents an action with authorization subject* as *and authorization object* ao. *Where* $z, newz \in U \cup H, u, u_1, u_2 \in U, d \in \Gamma, h, h_1, h_2 \in H, l \in H \cup \Gamma, m, prog, file, f, data \in O$, *and* M, P *are process specifications,*

$\boldsymbol{Enroll}_{z,d} \triangleq\ newz[\text{in } d.z[]] \mid d[\text{open } newz].0$

$\boldsymbol{Login}_{u,l} \triangleq\ u[\text{in } l] \mid l[]$

$\boldsymbol{Logout}_{u,l} \triangleq\ l[u[\text{out } l]]$

$\boldsymbol{Send}_{u_1,m} \triangleq\ d[h_1[u_1[m[M]|\text{out } u_1.\text{out } h_1.\text{in } h_2.\text{in } u_2.0]]]|[h_2[u_2[]]]]$

$\boldsymbol{Receive}_{u,m} \triangleq\ u[\text{open } m.(m)|m[M]]$

$\boldsymbol{Execute}_{u,prog} \triangleq\ d[h[u[\text{open } prog] \mid prog\ [\text{in } u.P]]]$

$\boldsymbol{Read}_{u,file} \triangleq\ h[file[data[\text{in } u.0]] \mid u[\text{in } file.0]]$

$\boldsymbol{Write}_{u,file} \triangleq\ h[file[T] \mid u[\text{in } file.data[\text{out } u.0]]]$

$\boldsymbol{Delete}_{u,file} \triangleq\ h[file[\text{in } u] \mid u[\text{open } file.0]]]$

$\boldsymbol{Create}_{u,file} \triangleq\ h[u[\text{open } f.f[file[\text{out } f.0]]]]$

Based on Definition 4.10, we present some examples of object, host and user mobility specification of mobility as Ambient Calculus processes. Some known notation conventions are utilized, for example n[] means n[0]. The symbol $\rightarrow$ represents the reduction relation and $\rightarrow^*$ represents a series of reductions.

- Object Mobility: File f_1 is moved from directory dr_1 in host h_1 to directory dr_2 in host h_2.

$d_1[h_1[dr_1[out\ dr_1.out\ h_1.in\ h_2.in\ dr_2.f_1[]]]|h_2[dr_2[]]] \rightarrow^* d_1[h_1[dr_1[]]|h_2[dr_2[f_1[]]]]$

- Host Mobility: Portable host h_1 moves from Domain d_1 to Domain d_2.
  $d_1[h_1[out\ d_1.in\ d_2.0]]|d_2[] \rightarrow^* d_1[]|d_2[h_1[]]$
- User Mobility: User u_1 logs into Host h_1, which contains File f_1.
  $u_1[in\ h_1.0]|h_1[f_1[]] \rightarrow h_1[u_1[]|f_1[]]$

4.3.3. Location Configuration

The location configuration of the system LCONF holds the formal specification for the current system. The initial state, $LCONF_0$ is derived from the network configuration and security policy specification. When an action is requested by an user, the action is matched to its formal representation, and if it is allowed by the security policy, the action is executed and the location configuration is updated.

In the formal specification, domains, hosts, users and objects are modeled as Ambients. The actions in the security policy rules are modeled using Ambient Calculus capabilities. A process specification shows a trace of a process in a certain mobile network location configuration. Each location configuration may be modeled as a process specification. This specification will then be checked against a location formula for compliance. The process specification involves capabilities, objects and ambients. A capability represents the potential to execute an action. As an example process specification, $in\ c.(R).T$ means that the ambient running the process will enter ambient c, input the object R and continue execution as process T.

Figure 4.5 is a representation of the Ambient Calculus based mobile process specification: $a[b|c[< R > .W|R]|d[in\ c.(R).T]]$. A snapshot of a mobile process is represented with a tree. Each action execution results in the modification of a tree. In the figure the execution of *in* action is shown. The locations are represented by tree nodes. Each non-leaf node holds an ambient as a child node. Leaf nodes contain process definitions and objects (resources). Resources may be *input* and *output* by the ambients. The ambients may represent World, Domains, Hosts and Users.

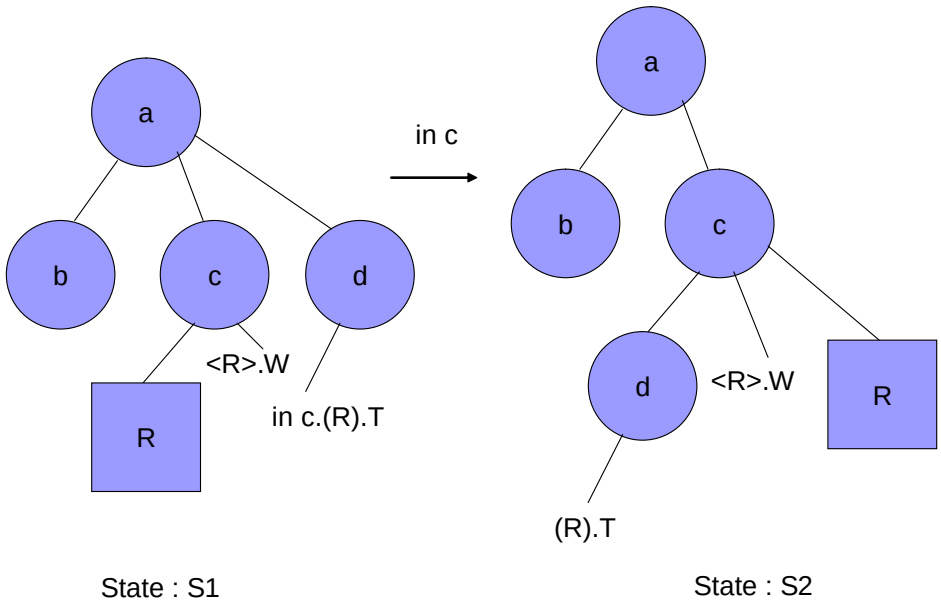

Figure 4.5. Representation of a location configuration in the form of a location hierarchy.

An example mobile network specification in Ambient Calculus is as follows: $d_1[u_1[in\ h_1.0|out\ h_1.0|in\ f_1.0|out\ f_1.0]|h_1[f_1[data_1[in\ u_1.0|out\ u_1.0]]]]$. In this example there is one domain, *d1*, user *u1* has permission to read data from file *f1* by logging into host *h1*. The actions in the security policy rules are formalized as Ambient Calculus capabilities. Multiple actions are combined with the parallel (|) operator since any of these actions may be exercised independently.

An example mobile network specification in Ambient Calculus which combines multiple actions is presented below. In this example there are two domains, d_1 and d_2. Mobile user u_2 has the right to move into either of the two domains, login to h_1 and h_2 and read files f_1 and f_2 . The formal representation of actions in multiple security policy rules are combined with the parallel (|) operator since any of these actions may be exercised independent from each other.

$$LCONF = d_1[u_1[]|h_1[f_1[data_1[in\ u_2.0|out\ u_2.0]]]]|d_2[h_2[u_2[out\ h_2.0|out\ d_2.0$$
$$|in\ d_1.in\ h_1.0|out\ h_1.out\ d_1.0|in\ d_2.in\ h_2.0|in\ f_1.0|in\ f_2.0|out\ f_1.0|out\ f_2.0]|f_2[data_2[]]]]$$

The location configuration changes with respect to the state changes in the sys-

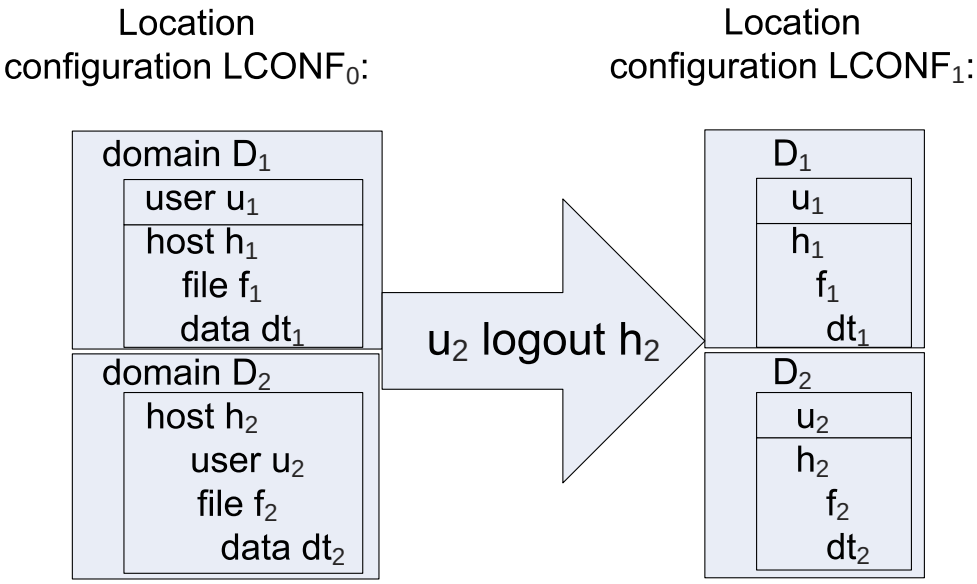

Figure 4.6. Example for change of location configuration with actions.

tem. In the initial state, $LCONF_0$ is derived from the network configuration and security policy specification. The domains, users, hosts and objects in the system and their locations are derived from defined data sets. The set of possible actions are derived from the security policy. When an action is requested by a user, the action is matched to its formal representation according to Definition 4.10, and if allowed by the security policy, it is executed and the location configuration is updated accordingly. The state change between different location configurations takes place with a sequence of Ambient Calculus capabilities corresponding to actions. Here is an example for modifying the location configuration with actions. The location configuration of the system at time T_0 and T_1 are depicted in Figure 4.6. At the initial state the Ambient Calculus specification is as follows:

$$LCONF_0 = D_1[u_1[]|h_1[f_1[dt_1[in\ u_2.0|out\ u_2.0]]]]|D_2[h_2[u_2[out\ h_2.0|out\ D_2.0|$$
$$in\ D_1.in\ h_1.0|out\ h_1.out\ D_1.0|in\ D_2.in\ h_2.0|in\ f_1.0|in\ f_2.0|out\ f_1.0|out\ f_2.0]|f_2[dt_2[]]]]$$

In Step (1), User u_2 logs out of Host h_2 corresponding to Ambient Calculus capability $u_2[out\ h_2...]$:

$$LCONF_1 = D_1[u_1[]|h_1[f_1[dt_1[in\ u_2.0|out\ u_2.0]]]]|D_2[u_2[out\ D_2.0|in\ D_1.in\ h_1.0|$$

Table 4.6. Location and mobility constraints in the security policy rule.

Constraint	**Represents**	**Represented by**
Location	Locations of objects, users, hosts and domains	Somewhere ($\diamond$) modality, parallel ($\mid$) and ambient ($[]$) formalizations of Ambient Logic
Mobility	The change of locations with time	Sometime ($\diamondsuit$) and Everytime ($\Box$) modalities in Ambient Logic

$out\ h_1.out\ D_1.0|in\ D_2.in\ h_2.0|in\ f_1.0|in\ f_2.0|out\ f_1.0|out\ f_2.0]|h_2[f_2[dt_2[]]]]$

4.3.4. Location and mobility constraints in security policy rules

The location constraints in the security policy rules are specified using Ambient Logic. *Location formula* (fo) which represents location and mobility constraints is a modal logic formula with spatial and temporal constructs. It is a formula in spatial logic that includes names of domains, authorization subjects and authorization objects. Ambient Logic is used since it is possible to represent both time and location. We note here that location in our model is a logical concept that shows the relative location of objects and users to hosts and domains rather than their physical location. The use of formula in the policy rule is explained in Table 4.6.

If the location formula contains the Somewhere ($\diamond$), parallel ($\mid$) and ambient ($[]$) formalizations of Ambient Logic, it is referred as a *location constraint*. Additionally, if it contains temporal constructs which are Sometime ($\diamondsuit$) and Everytime ($\Box$) modalities in ambient logic, it is referred as a *mobility constraint*. Location constraints are evaluated by spatial model checking, whereas mobility constraints are evaluated by a further step of temporal model checking.

As an example to location and mobility constraints, we present a location formula as follows: $fo = \Box\{\neg\diamondsuit\{\diamond d_2[\ \diamond\{data1[\top]|data2[\top]\}]|\top\}\}$. This is an Ambient Logic specification stating that Domain d_2 should never contain *data1* and *data2* at the same time. If combined with specification *LCONF* in the previous section which specifies

that *data1* resides in d_1 and *data2* resides in h_2 inside d_2, the interpretation of this mobility constraint is that domain d_2 data should not be copied to domain d_1.

4.4. Formal Specification of Security Policy Rules

The formal specification of security policy rules is based on predicate logic and ambient modal logic within the authorization term structure presented in Section 4.1.8. Here, we present the formalization process through realistic security policy examples. The policy statement in verbal form is followed the corresponding formally specified authorization term. There are three types of policy specification. The first one is generic policy statements valid for all network models. The second one is domain security policy statements defined by a system or security administrator specific to a network. The third one is inter-domain security policy statements defined by a home domain security administrator for interactions among home and foreign domains.

4.4.1. Generic Security Policy Rules

First, we list examples of generic policy statements that can be defined by following our method. In these generic statements, $host \in H$, $object \in O$, $user \in U$, $domain \in \Gamma$.

- "Users must be enrolled to a domain before they can login.":
  (as = user, ao = domain, sa = login, $fo = \diamond(domain[T]|user[T]|T)$, $co = \forall user \in U, \exists domain \in \Gamma, EDR(domain, user)$)
- "Users must be logged into a host before any other action can be conducted by a user in a domain.":
  (as = user, ao = domain, sa = A\{login}, $fo = \diamond(domain[host[user[T]]|T]|T)$, $co = \forall user \in U, \exists domain \in \Gamma, \exists host \in H$)
- "Users can conduct an action on a host except login only if already logged into that host.":
  (as = user, ao = host, sa = A\{login}, $fo = \diamond(host[user[T]|T]|T)$, $co = \forall user \in U, \exists host \in H$)

- "Users can not conduct any action on an object contained in a host if the user is not logged into that host.":
  $(as =$ user, $ao =$ object, $sa = \emptyset$, $fo = \diamond(host[object[T]|T]|user[T]|T)$, $co = \forall user \in U, \exists object \in O, \exists host \in H)$
- "Hosts must be connected to a domain by a user of that domain before any other action can be conducted by a host.":
  $(as =$ user, $ao =$ host, $sa =$ A\{connect}, $fo = \diamond(domain[T]|host[T]|T)$, $co = EDH(domain, host) \wedge EDR(user, domain))$
- "Hosts must be enrolled to a domain by an administrator before they can be connected.":
  $(as =$ user, $ao =$ host, $sa =$ connect, $fo = \diamond(domain[T]|host[T]|T)$, $co = \forall host \in H, \exists domain \in \Gamma, EDH(domain, host) \wedge ADM(user, domain))$
- "Enrolling any entity to a domain can be done by a user with administrative rights.":
  $(as =$ user, $ao =$ domain, $sa =$ enrol, $fo =$T, $co = \forall domain \in \Gamma, \exists user \in U, ADM(user, domain))$

4.4.2. Formal Specification of Security Policy Rules with Location and Mobility Constraints for Domain Security Policies

Below are some examples for formal specification of domain security policy statements with location and mobility constraints. In this specification example, we assume the following: $UniA \in \Gamma$, $\{jfrantz, nmullis, cmiele\} \subset \mathbf{U}$, $\{h_{11}, h_{12}, h_{15}\} \subset \mathbf{H}$, $\{Portable, Message, File, Project\_Folder\} \subset \mathbf{T}$.

- "All users can read files in folder *Project_Folder*, if they are in a location that contains this folder": (as = user, ao = *Project_Folder*, sa = + *read*, fo = $\diamond(as[] \mid ao[])$, co = $user \in U \wedge OIT(ao, Project\_Folder)$)
- "Files in Host h_{11} can not be read by any user from portable hosts.": $(as =$ user, $ao =$ File, $sa =$ (-) read, $fo = \diamond(host[user[T]|T]|h_{11}[file[]])$,
  $co = \forall user \in U, (OIT(file, File) \wedge OIT(host, Portable))$
- "User *nmullis* may send messages to User *jfrantz*".: $(as =$ nmullis, $ao =$ Message,

$sa =$ send, $fo = \diamond(nmullis[m[]|T]) \wedge \diamond(jfrantz[m[]]|T))$, $co = OIT(m, Message))$

- "Portable host h_{15} can be connected to Domain UniA by users of UniA".: ($as =$ user, $ao = h_{15}$, $sa =$ connect, $fo = \diamond(h_{15}[T]|UniA[T])$, $co = \exists user \in U, \exists h_{15} \in H, \exists UniA \in \Gamma, EDR(user, UniA) \wedge OIT(h_{15}, Portable))$
- "User *cmiele* can login to Host h_{12}" : ($as =$ cmiele, $ao = h_{12}$, $sa =$ login, $fo = \diamond(h_{12}[T]|cmiele[T])$, $co =$ T)

4.4.3. Formal Specification of Security Policy Rules with Location and Mobility Constraints for Inter-Domain Security Policies

In this section, we present examples for formal specification of inter-domain security policy rules. The formal specification includes the mobility and access control aspects of the inter-domain policy. Example policy statements for mobility and inter-domain access as a verbal form of policy rule are given followed by formal specification of authorization terms and an interpretation of the location and mobility constraints. For this specification we assume the following:

$\{UniA, UniB\} \subset \Gamma, \{Student, Lecturer, Researcher\} \subset R, \{portable, prj\_srv\} \subset O, Portable \in T$, $\mathbf{RH} = \{(Member, Student), (Member, Lecturer)\}$,
$\mathbf{OTH} = \{(Object, Host), (Host, Portable)\}$.

- "Students of University B are allowed to connect their portables to University A.":
  ($as =$ Student, $ao =$ portable, $sa =$connect,$fo = World[UniB[T]|portable[as[T]|T]|UniA[T]|T]$,$co = (EDR(as, UniB) \wedge RAS(as, Student) \wedge OIT(portable, Portable))$
- "Students of University B, once logged on to the University A domain, can not login to the project server.":
  (as=Student, ao=prj_srv, sa=(-)login, fo=$World[UniA[\diamond(as[T]|prj\_srv[T])|T]|UniB[T]|T]$, co = $(EDR(as, UniB) \wedge ADU(as, UniA) \wedge RAS(as, Student))$
- "Lecturers of University B can not login to the project server in University A domain from within University B domain.":

(as = Lecturer, ao = prj_srv, sa = (-) login, $fo = World[UniA[prj\_srv[T]|T]|UniB[\diamond as[]]|T]$, $co = EDR(as, UniB) \wedge ADU(as, UniB) \wedge RAS(as, Lecturer)$)

- "Researchers of University B can read data from files on the project server in University A domain from within University A.":
  (as = Researcher, ao = prj_srv, sa = read, $fo = World[UniA[prj\_srv[file[data]|T]|\diamond as[]|T]|T]$, $co = EDR(as, UniB) \wedge ADU(as, UniA) \wedge RAS(as, Researcher)$
- "Lecturers of University B with a Portable host can not write files to the project servers of University A.":
  (as = Lecturer,ao= prj_srv, sa = (-) write,$fo = World[UniA[portable[as[file[]]]|project\_server[]|T]|T]|T]$, $co = EDR(as, UniB) \wedge ADU(as, UniA) \wedge RAS(as, Lecturer) \wedge OIT(portable, Portable)$)

4.5. Separation of Duty (SOD) Constraints in FPM-RBAC

In FPM-RBAC, novel classes of SOD constraints are identified: (i) Service based SOD, (ii) Inter-Domain SOD and (iii) Location and Mobility Based SOD. Service-based SOD is defined on the basis of a set of conflicting services CS. CS is a set of services, where two or more of its members may not be assigned to a single role. For example, a role may not be assigned to services "Secret" and "Unclassified". Inter-Domain SOD may be based on inter-domain role assignment, home role mapping or foreign role mapping. Location based SOD may be based on service location, role location or permission location. The classification of SOD in FPM-RBAC is presented in Figure 4.7. Static or dynamic aspects are represented by time of evaluation. Static SOD is evaluated statically against initial system state, while Dynamic SOD is evaluated at runtime against current system state.

In [66], the SOD constraints are specified within the Role-Based Constraints Language using the sets of conflicting roles, permissions and user sets in addition to the standard RBAC model. Mutually disjoint roles are defined as conflicting roles set CR. The concept of conflicting permissions defines conflict in terms of permissions rather than roles. Two permissions may be considered in conflict independent from

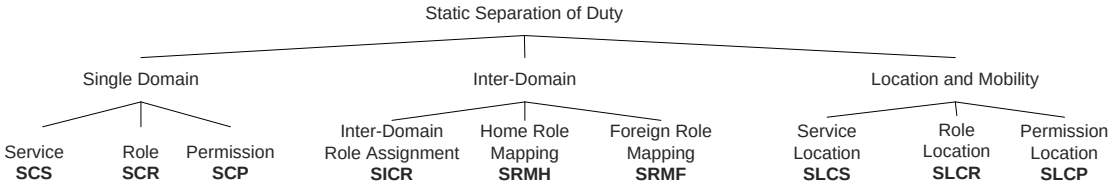

Figure 4.7. Classification of static separation of duty in FPM-RBAC.

role assignment relation. For example, permissions to write grades to student files and to approve grades of students may be declared as conflicting permissions. Role and Permission based SOD are included as Single-Domain constraints (SCR and SCP) in this classification. An interplay of the three classes of SOD constraints are also possible however a detailed analysis is out of scope of this study.

4.5.1. Single Domain SOD

The Single Domain SOD is defined based on the static separation of duty definition in the standard RBAC model. We differentiate between role-based, permission-based and service-based SOD in FPM-RBAC model. Service Based SOD is a novel concept defined in the FPM-RBAC model. The sets of roles, permissions and services which are the basis for mutual exclusion are defined as CR (conflicting roles), CP (conflicting permissions) and CS (conflicting services) respectively. The Static Separation of Duty based on roles follows the standard RBAC model definitions [55]. Additionally in FPM-RBAC, SCS (SOD based on Conflicting Services) and SCP (SOD based on Conflicting Permissions) are defined as follows:

4.5.1.1. Static Separation of Duty based on Roles. $SCR \subseteq (2^R \times N)$ is a collection of pairs where $CR = \{r_x, \ldots, r_y : r_x, \ldots, r_y \in R\}$ is a set of conflicting roles, t a subset of roles in CR, and n a natural number $n \geq 2$, s.t. no user is assigned to n or more roles from the set CR in each $(t, n) \in SCR$. SCR is defined in Equation 4.1. In presence of a hierarchy, the function *assigned_users (r)* in Equation 4.1 is replaced with the

function *authorized_users (r)* to yield Equation 4.2.

$$\forall(t,n) \in SCR, \forall t \subseteq CR :| \ t \ | \geq n \rightarrow \bigcap_{r \in t} assigned\_users(r) = \emptyset \tag{4.1}$$

$$\forall(CR,n) \in SCR^*, \forall t \subseteq CR :| \ t \ | \geq n \rightarrow \bigcap_{r \in t} authorized\_users(r) = \emptyset \tag{4.2}$$

4.5.1.2. Static Separation of Duty based on Services. $SCS \subseteq (2^V \times N)$ is a collection of pairs where $CS = \{v_x, \ldots, v_y : v_x, \ldots, v_y \in \mathcal{V}\}$ is a set of conflicting services, s a subset of services in CS, and n a natural number $n \geq 2$, s.t. no *role* is assigned to n or more services from the set CS in each $(s,n) \in SCS$. *SCS* is defined in Equation 4.3. In presence of a hierarchy, the function *enabled_roles (v)* in Equation 4.3 is replaced with the function $enabled\_roles^*(v)$ to yield Equation 4.4.

$$\forall(s,n) \in SCS, \forall s \subseteq CS :| \ s \ | \geq n \rightarrow \bigcap_{v \in s} enabled\_roles(v) = \emptyset \tag{4.3}$$

$$\forall(CS,n) \in SCS^*, \forall s \subseteq CS :| \ s \ | \geq n \rightarrow \bigcap_{v \in s} enabled\_roles^*(v) = \emptyset \tag{4.4}$$

4.5.1.3. Static Separation of Duty based on Permissions. Where $CP = \{cp_i : cp_i \in ACT \times AO\}$ is a set of conflicting permissions, the set of conflicting inter-domain roles is interpreted as $CR = \{r_x : permissions^*(r_x) \cap cp_i \neq \emptyset\}$. Equation 4.1 is applicable with the derived CR set.

4.5.2. Inter-Domain SOD

The introduction of role maps arise additional concerns about separation of duty. Because of multiple sets of conflicting roles for home, foreign and inter-domain roles, new types of SOD need to be defined. First we define the set of conflicting inter-domain roles, $CR_\Gamma = \{r^x_\Gamma, \ldots, r^y_\Gamma : r^x_\Gamma, \ldots, r^y_\Gamma \in R_\Gamma\}$. We identify the following types of static SOD related with inter-domain role mapping:

(i) $SICR$: No user can be assigned to a set of roles which map to n or more conflicting inter-domain roles.

(ii) SRM_h: No user can be assigned by mapping to n or more inter-domain roles which are mapped by a set of conflicting home domain roles.

(iii) SRM_f: No user can be assigned by mapping to n or more inter-domain roles which are mapped by a set of conflicting foreign domain roles.

We introduce a function $rmap(r) : R \times R_\Gamma$ which gives the set of mapped inter-domain roles for a set of home or foreign domain roles. The formal definitions for the newly introduced static SOD constraints $SICR, SRM_h, SRM_f$ are defined in Equations 4.5, 4.6 and 4.7.

$$\forall(\alpha, n) \in SICR, \forall\alpha \subset R, |rmap(\alpha)| \geq n, \tag{4.5}$$
$$\bigcap_{r\in\alpha} assigned\_users(r) \neq \emptyset \rightarrow rmap(\alpha) \nsubseteq CR_\Gamma$$

$$\forall(\alpha, n) \in SRM_h, \forall\alpha \subset R, |rmap(\alpha)| \geq n, \tag{4.6}$$
$$\bigcap_{r\in\alpha} assigned\_users(rmap(r)) \neq \emptyset \rightarrow \alpha \nsubseteq CR_h$$

$$\forall(\alpha, n) \in SRM_f, \forall\alpha \subset R, |rmap(\alpha)| \geq n, \tag{4.7}$$
$$\bigcap_{r\in\alpha} assigned\_users(rmap(r)) \neq \emptyset \rightarrow \alpha \nsubseteq CR_f$$

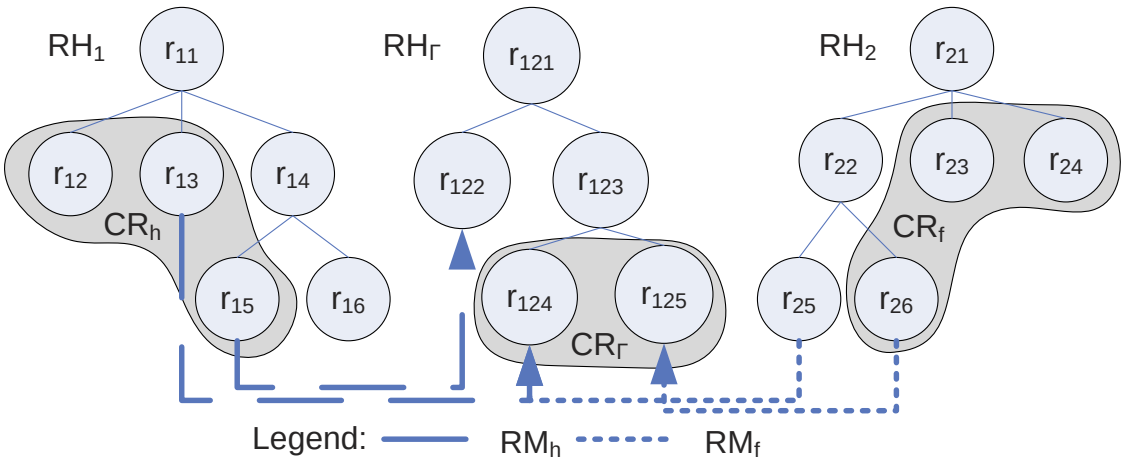

Figure 4.8. Conflicting role sets for inter-domain hierarchies.

As an example of static SOD relations for inter-domain role mapping, consider the example depicted in Figure 4.8. Let $CR_h = \{r_{12}, r_{13}, r_{15}\}, CR_\Gamma = \{r_{124}, r_{125}\}, CR_f = \{r_{23}, r_{24}, r_{26}\}$. The home domain mapping is $RM_h = \{(r_{13}, r_{124}), (r_{15}, r_{122})\}$. The foreign domain mapping is $RM_f = \{(r_{25}, r_{124}), (r_{26}, r_{125})\}$. The role assignment $\{(u, r_{122}), (u, r_{124})\}$ is conflicting according to SRM_h and the role assignment $\{(u, r_{25}), (u, r_{26})\}$ is conflicting according to *SICR*.

SOD relations may be defined in an alternative way. One may define sets of conflicting permissions instead of conflicting roles. In this case, the conflicting roles can be interpreted as the roles that have conflicting permissions assigned to them. In this case, the set of conflicting inter-domain permissions can be defined by the administrator as $CP_\Gamma = \{cp_i : cp_i \in ACT \times AO\}$ and the set of conflicting inter-domain roles is interpreted by the system as $CR_\Gamma = \{r_\Gamma^x : permissions^*(r_\Gamma^x) \cap cp_i \neq \emptyset\}$. The inter-domain SOD constraints presented in this section are then applicable with the derived CR_Γ set.

4.5.3. SOD for Location and Mobility Constraints

We identify new aspects about the specification of SOD constraints together with location and mobility constraints. Two or more roles or permissions may be considered in conflict if one of these permissions relates to an action which involves mobility. For example, the *Guest Lecturer* role may be in conflict with the *Lecturer* role if the person is visiting another university. As an example to permission conflicts arising from mobility, consider a requests to send a student record file to another institution and at the same time to change the records. In this case, the records which have been sent would be in conflict with the ones changed.

In the FPM-RBAC model, Static and Dynamic SOD constraints may be used in accordance with Location and Mobility Constraints. The generic form of location and mobility based SOD constraints based on set $\mathcal{C} \in \{CR, CP, CS\}$ is $(\mathcal{C}, n, sfo)$ where n is a natural number and *sfo* is a (set of) location constraint(s) for separation of duty. Set $\mathcal{C}$ is specified depending on whether the constraint is role-based, permission-based

or service-based. We associate the SOD constraints with the Location and Mobility constraints for conflicting roles and conflicting services as follows:

(i) Conflicting Roles with Location Constraints (SLCR): n or more roles may not be assigned to a user if these roles are within the conflicting roles set CR and if the Location constraint *sfo* is valid in the initial location configuration of the system.

(ii) Conflicting Services with Location Constraints (SLCS): n or more services from the set of conflicting services CS may not be assigned to a role if the location constraint *sfo* is valid in the initial location configuration of the system.

Static location constraints are evaluated against the initial location configuration ($LCONF_0$), which depicts the state of the network before any action has been executed. The formal definitions of $SLCR$, location and mobility SOD based on conflicting roles, and $SLCS$, location and mobility SOD based on conflicting services, are defined in Equations 4.8, 4.9, 4.10 and 4.11. In the presence of hierarchies, $SLCR^*$ is obtained by replacing the function *assigned_users* in $SLCR$ with the function *authorized_users*. Similarly, $SLCS^*$ in the presence of hierarchies is obtained by replacing *enabled_roles(r)* in $SLCS^*$ with $enabled\_roles^*(r)$.

$$SLCR(n, sfo) = \forall (t, n) \in SLCR, \forall t \subseteq CR :| \ t \ | \geq n \rightarrow \bigcap_{r \in t} assigned\_users(r) = \emptyset \wedge LCONF_0 \vDash sfo \tag{4.8}$$

$$SLCS(n, sfo) = \forall (s, n) \in SLCS, \forall s \subseteq CS :| \ s \ | \geq n \rightarrow \bigcap_{v \in s} enabled\_roles(v) = \emptyset \wedge LCONF_0 \vDash sfo \tag{4.9}$$

$$SLCR^*(n, sfo) = \forall (CR, n) \in SLCR^*, \forall t \subseteq CR :| \ t \ | \geq n \rightarrow \bigcap_{r \in t} authorized\_users(r) = \emptyset \wedge LCONF_0 \vDash sfo \tag{4.10}$$

$$SLCS^*(n, sfo) = \forall (CS, n) \in SLCS^*, \forall s \subseteq CS :| \ s \ | \geq n \rightarrow \bigcap_{v \in s} enabled\_roles^*(v) = \emptyset \wedge LCONF_0 \vDash sfo \tag{4.11}$$

Location and mobility based SOD constraints based on conflicting permissions are specified in a different way. The reason is that the authorization term associates permissions with location constraints. A number n or more authorization terms are in conflict if among all authorization terms assigned to the same authorization subject, n or more permissions are within the conflicting permissions set $CP = \{cp_i : cp_i \in ACT \times AO\}$ and if the Location constraints associated with the permissions are all valid in the initial location configuration of the system. In this case the location formula for separation of duty is defined by the set $\{fo_1, \ldots fo_n\}$ associated with authorization terms $\{at_1, \ldots at_n\}$. The formal definition for $SLCP$, SOD based on Conflicting Permissions with Location Constraints, is given in Equation 4.12.

$$\begin{gathered} \forall (p, n) \in SLCP, \forall \alpha \subseteq AT, \forall p \subseteq CP :\mid p \mid \geq n, \\ \alpha = \{(as, p_1, fo_1, co_1), \ldots (as, p_n, fo_n, co_n)\}, \\ permissions(as) \subseteq p \rightarrow LCONF_0 \vDash \{fo_1, \ldots fo_n\} \end{gathered} \tag{4.12}$$

In presence of an hierarchy, the assignment of permissions to authorization subjects are evaluated according to the role hierarchy. In this case, the function $permissions(as)$ in Equation 4.12 is replaced by the function $permissions^*(as)$.

4.5.4. Dynamic SOD

Here we define the dynamic SOD constraints in the FPM-RBAC model. Similar to static SOD constraints, dynamic SOD constraints also have three different types, namely, role based, service based and location/mobility based dynamic SOD. The main differentiation between Static and Dynamic SOD constraints are, (i) dynamic constraints are based on activation of roles, which occurs when a user logs into a service, rather than assignment or roles and (ii) dynamic constraints are evaluated at run-time against the current state of the system, in which a set of actions has already been executed. The spatial and temporal location and mobility constraints enable the specification of complex temporal and spatial restrictions on separation of duty. The concept of services, which binds a dynamic login session to a set of authorization

objects, replaces the concept of sessions in standard RBAC. Since the user may login to one service at a time, the DSOD definition based on services is related to the set of enabled roles within a service.

4.5.4.1. Dynamic Separation of Duty based on Roles. $DCR \subseteq (2^R \times N)$ is a collection of pairs where $CR = \{r_x, \ldots, r_y : r_x, \ldots, r_y \in R\}$ is a set of conflicting roles, t a subset of roles in CR, and n a natural number $n \geq 2$, s.t. no user has actively assumed n or more roles from the set CR in each $(t, n) \in DCR$. The formal definition of DCR is given in Equation 4.13.

$$\forall (t, n) \in DCR, \forall u \in U, \forall t \subseteq CR :\mid t \mid \geq n \rightarrow \bigwedge_{r \in t} RAS(u, r) = \bot \tag{4.13}$$

4.5.4.2. Dynamic Separation of Duty based on Services:. $DCS \subseteq (2^R \times N)$ is a collection of pairs where $CR = \{r_x, \ldots, r_y : r_x, \ldots, r_y \in R\}$ is a set of conflicting roles, t a subset of roles in CR, and n a natural number $n \geq 2$, s.t. no user, when logged into a service $v \in \mathcal{V}$, may activate n or more roles from the set CR in each $(t, n) \in DCS$. The formal definition of DCS is given in Equation 4.14.

$$\begin{gathered} \forall (t, n) \in DCS, CR \subset 2^R, \forall t \subseteq CR, \forall v \in \mathcal{V}, \\ t \subset enabled\_roles(v) \wedge \mid t \mid \geq n \rightarrow \bigwedge_{r \in t} REN(v, r) = \bot \end{gathered} \tag{4.14}$$

4.5.4.3. Dynamic Separation of Duty based on Locations and Mobility. Location and mobility based dynamic SOD constraints are evaluated against the location configuration of the system at time τ ($LCONF_\tau$), after a set of actions has already been executed by users. The dynamic constraints take hierarchies into account since the predicates RAS and REN take into account the effect of descendant roles. Where $CR = \{r_x, \ldots, r_y : r_x, \ldots, r_y \in R\}$ is a set of conflicting roles, the formal definitions of dynamic location and mobility SOD constraints based on conflicting roles $DLCR$ and

conflicting services $DLCS$ are given in Equations 4.15 and 4.16.

$$\forall(t, n, \tau, sfo) \in DLCR, \forall u \in U, \forall t \subseteq CR, \tag{4.15}$$
$$\mid t \mid \geq n \rightarrow \bigwedge_{r \in t} RAS(u, r) = \bot \wedge LCONF_{\tau} \vDash sfo$$

$$\forall(t, n, \tau, sfo) \in DLCS, CR \subset 2^{R}, \forall t \subseteq CR, \forall v \in \mathcal{V}, \tag{4.16}$$
$$t \subset enabled\_roles(v) \wedge \mid t \mid \geq n \rightarrow \bigwedge_{r \in t} REN(v, r) = \bot \wedge LCONF_{\tau} \vDash sfo$$

4.6. Authorization of Access Requests According to FPM-RBAC Security Policies

The algorithm presented in this section is utilized for making the permission or denial decisions for requested actions against security policy specifications. Determination of whether the actions in the multi-domain mobile network are permitted requires a formal linkage of access requests to the security policy model. Checking the satisfaction of current state of the system to constraints in the security policy is essential to match rules to policies. The location configuration of the system LCONF holds the Ambient Calculus process specification for the current system. With each action this configuration is updated. The location constraints which are enabled in the current location configuration of the system is determined by spatial model checking. This computation is decidable when using the fragments of Ambient Calculus and logic presented in this paper. The set of functions and predicates in domain policies and the inter-domain policies are defined respectively by the set of security policies Ω and $\mathcal{W}$. The state of the logical functions and predicates are tracked by the system. The Generic constraints as well as SOD constraints are evaluated according to the current state of the logical functions and predicates.

Matching actions to policy rules requires the following steps:

(i) Determine the applicable authorization terms based on the service activated by the user,

(ii) Map the authorization subjects in the policy rule to roles in the access request through role hierarchy definitions,

(iii) Map the authorization objects in the policy rule to the object on which an access is requested, through object hierarchy definitions,

(iv) Determine whether the location and mobility constraints are satisfied by the location configuration.

(v) Determine whether the Generic constraints and SOD constraints are satisfied.

(vi) Determine whether the action is permitted or denied through the Access Control Function.

The steps for checking the satisfaction of Location constraints and Generic constraints may be computed off-line during state changes. The evaluation of the Access Control function needs to be computed at the time of the request. In the following paragraphs, we investigate the functions and algorithms necessary for the execution of the steps mentioned above.

4.6.1. Evaluation of Access Requests according to Services

An access request ar is specified as $ar = \langle u, serv, r_u, l_r, act, obj \rangle$ where $serv \in \mathcal{V} \cup \mathcal{I}_\Gamma, r_u \in R, l_r \in H \cup \Gamma, act \in A, obj \in O$. Here, $serv$ denotes the active service that the user has activated including inter-domain services, r_u is the role assumed by the user, l_r is the location of the user (a host or domain), act is the requested action and obj is the object upon which an action is requested.

The subset of domain security policy for domain i pertaining to the service activated by the user,$P_i^{'}$, is specified as $P_i^{'} \subset P_i : \{\forall v \in \mathcal{V}_i, serv = v\}$. Given an authorization term in of the form $at = (as, ao, sa, co, fo)$, the set of available authorization terms to a role within a service $AT_i^{'} \subset AT_i$ is specified in Equation 4.17.

$$AT_i^{'} = \bigcup_{at \in AT} RAS(as, r_u) \wedge REN(r_u, serv) \tag{4.17}$$

The calculation is straightforward since the permission assignment matrix already

defines permissions for each service. The authorization terms are the rules in the permission access matrix for the service which relate to the enabled roles.

4.6.2. Evaluation of Hierarchies

First, the system checks whether the requested role in the access request may be assumed by the user. This is equivalent to $RAS(u, r_u) = \top$. Second, the set of authorization terms which apply to the role and object hierarchies defined in the security policy needs to be calculated. This takes four steps, to find the set of authorization terms $at = (as, ao, sa, co, fo), at \in AT_i^{'}$, where:

(i) the user specified in the authorization term (if any) is equal to the user requesting the service, $AT_i^u = \{at|\ at \in AT_i^{'}, as = u\}$

(ii) the role specified in the authorization term is equal or a descendant role of the role specified in the access request, $AT_i^r = \{at|\ at \in AT_i^{'}, as = r_u \vee as \prec r_u\}$

(iii) the object specified in the authorization term is equal to the object upon which the access is requested,$AT_i^o = \{at|\ at \in AT_i^{'}, ao = obj\}$

(iv) the object specified in the access request is of object type specified in the authorization term, or the object type of the object is a descendant of the object type specified in the authorization term, $AT_i^{ot} = \{at|\ at \in AT_i^{'}, OIT(obj, ao) \vee object\_type(obj) \prec ao\}$

The set of derived authorization terms in presence of role and object type hierarchies $AT^{''}$ is $AT^{''} = AT_i^u \cup AT_i^r \cup AT_i^o \cup AT_i^{ot}$.

4.6.3. Checking Satisfaction of Location and Mobility Constraints

To check location constraints in security policy, we apply model checking of Ambient Calculus specifications against Ambient Logic formulas. In our works [41] and [67], we present a spatial and temporal model checking algorithm for checking satisfaction of location and mobility constraints. The calculation of satisfaction relation for location configuration against location and mobility constraints is formalized as

follows: for each authorization term $(as, ao, sa, co, fo) \in AT''$, where $LCONF$ is an ambient process specification for the current location configuration and fo is an Ambient Logic formula within the authorization term, determine whether $LCONF \models fo$. The details of the algorithm for model checking of Ambient Calculus specifications are in Chapter 6.

The set of authorization terms applicable for the current access request is further reduced according to the satisfaction of location constraints by the current location configuration of the system. The set of derived authorization terms which satisfy location constraints is $AT^{loc} = \{(as, ao, sa, co, fo) \in AT'' \mid LCONF \models fo\}$.

4.6.4. Checking Generic and SOD Constraints Specified by Conditions

The method for checking conditions in the security policy rules is as follows: Where $PRED$ is the set of satisfied predicates in the current state of the system and co is a condition, for each authorization term $(as, ao, sa, co, fo) \in AT''$, determine whether $PRED \models co$.

Since Predicate Calculus is utilized for specification of predicates and conditions in the security policy rule, the satisfaction relation is a logical entailment problem. We utilize an automated theorem prover for specification and verification of conditions. A detailed description of theorem prover support is presented in Chapter 7. The calculation of SOD relations are presented in detail in Section 4.5.

The set of authorization terms applicable for the current access request, which satisfy conditions is specified as $AT^{co} = \{(as, ao, sa, co, fo) \in AT'' | PRED \models co\}$.

4.6.5. Evaluation of the Access Control Function

The access control function (ACF) matches a requested permission against the set of permissions assigned to roles in a certain location. The set $permissions^*(r_u)$, denoting permissions for a role, derived from the set of authorization terms $at =$

$(as, ao, sa, co, fo), at \in AT''$ evaluated against hierarchies, can be calculated as specified in (4.18).

$$permissions^*(r_u) = (obj, act) \in \bigcup_{(ao,sa)\in P} at \in AT'' \tag{4.18}$$

When an access request ar by a user u with role r_u in a specific location l_r to conduct action act on object obj is received, ACF (ar) may return two values, *allowed* or *denied*. The definitions of the access control function is as follows.

Definition 4.11. *Access Control Function ACF is:*

$ACF(ar) =$ allowed *if:* $(l_r \in serv) \wedge (AT'' \neq \emptyset) \wedge (AT^{co} \cup AT^{loc} \neq \emptyset) \rightarrow \exists(+, act) \in permissions^*(r_u)$

$ACF(ar) =$ denied *if:* $(l_r \notin serv) \vee (AT'' = \emptyset) \vee \exists(-, act) \in permissions^*(r_u)$

The interpretation of the access control function is, if the location of the user is within locations associated with the service, if there is any derived authorization terms according to evaluation of user name, role, role hierarchy, object or object hierarchy, if the location and separation-of-duty constraints are applicable to the set of derived authorization terms, then there is at least one rule with a permission that has the same object and action as those within the access request. The denial takes place if the request is placed from a location outside the scope of the service, there is no derived rule for the request, or there is at least one derived rule with a specific denial (negative sign). The order of evaluation of denials versus allowed accesses may depend on precedence of signs (denials take precedence, allowances take precedence).

4.7. Comparison of FPM-RBAC with Other RBAC Models

In this section we compare FPM-RBAC to other RBAC models. First we make a comparison between FPM-RBAC and the original RBAC model, which is a standard proposed by NIST. Second, we compare FPM-RBAC to extensions of the RBAC model

which cover spatial and temporal aspects.

4.7.1. Comparison of FPM-RBAC with NIST RBAC Model

The NIST Role-Based Access Control model proposed by [55] consists of five basic elements, Users, Roles, Operations and Objects. The NIST RBAC model is a single-domain model, where the concepts of domain and inter-domain policies are not differentiated. Roles are assigned to users with the UA (User Assignment) relation. Permissions are assigned to roles with the PA (Permission Assignment) relation. The activation of roles is based on the concept of Sessions. We use the constrained RBAC model that supports role hierarchies and separation of duty (SOD) relations as a basis for our policy model. The constrained RBAC model is depicted in Figure 4.9.

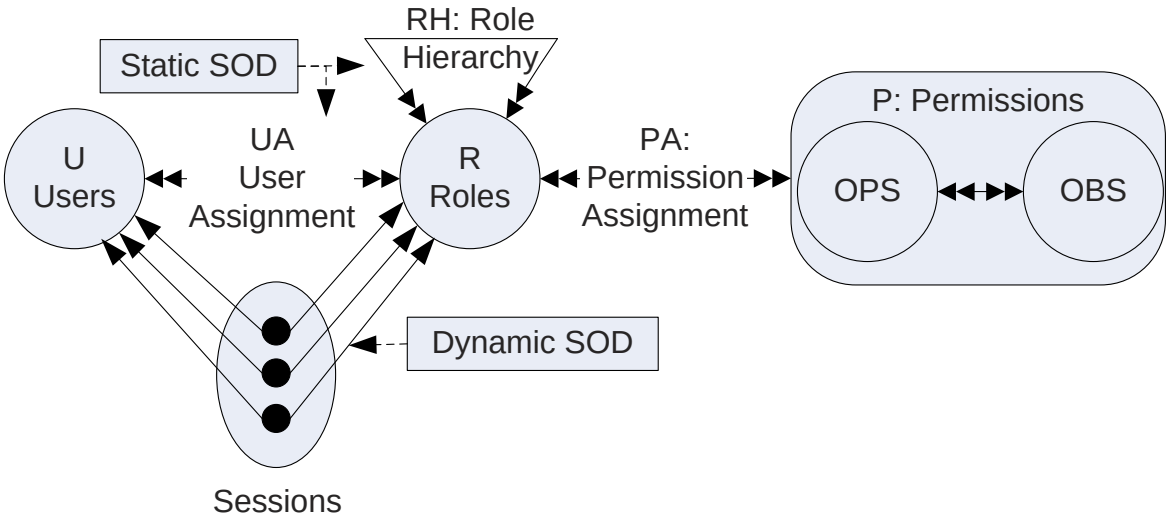

Figure 4.9. The NIST constrained RBAC model.

In contrast to NIST RBAC model, FPM-RBAC is a multi-domain access control model based on the concepts of *domain* and *inter-domain* policies. The domain policy model of FPM-RBAC introduces Location Constraints, Services and Object Type Hierarchy. The activation of roles in FPM-RBAC Access Control Model is achieved through the use of Services instead of Sessions. Each service is bound to a set of Authorization Objects, namely, Domains, Hosts, Object Types and Objects. The decision to introduce Services arises from the service-oriented nature of multi-domain networks with heterogeneous and interconnected services such as web services and

network services. The concept of session, which represents a dynamic binding of users to a set of roles, is not sufficient to express the use of services. A service restricts the user-role mapping to a set of authorization objects bound to the related service.

Another difference is the use of Object Type Hierarchy (OTH) in FPM-RBAC. Object-Type Hierarchies were introduced by Jajodia *et al.* [23] in the Flexible Authorization Framework (FAF). In a multi-domain setting, the knowledge of objects across domains is not desirable because of distributed administration. Through the use of Object Types the permissions may be mapped across multiple domains without the need of global knowledge of individual objects.

We also introduce SOD relations based on services, inter-domain role mapping, locations and mobility. The introduced SOD relation types in FPM-RBAC support specification of SOD relations for service-based multi-domain mobile networks.

4.7.2. Comparison of FPM-RBAC with Existing Spatial and Temporal RBAC Models

The recent studies on spatial and temporal RBAC models are more suitable for environments in which the location of users and objects are static in nature. The LoT-RBAC model does not include a formal link between the hierarchy model and the location relations. STARBAC does not have a formal spatial model, but rather uses a set of physical points in two dimensional space to define locations. The mapping relations supports determination of logical entailment for a spatial and temporal condition. However this model does not support more detailed analysis such as spatial model checking where a spatial condition needs to be matched to the spatial configuration of the environment. Locations in STARBAC are not hierarchical or mobile, the relationships between them should be defined explicitly. Using fixed locations is not ideal for mobile network environments.

In contrast with GEO-RBAC, FPM-RBAC model location formulas are used instead of extents, which are logical formulas based on the concept of ambients that

define logical locations. Role enabling is achieved through Service Access Matrix, where roles are given access to services. Role schemas are used in GEO-RBAC to define logical location boundaries for roles, whereas in FPM-RBAC they are defined as location constraints for User-Role assignment relation. Because of this decision, there is no need for a separate role schema hierarchy in FPM-RBAC. The allowed set of authorization terms for a given spatial configuration of the system is determined by spatial model checking. In GEO-RBAC this decision is computed by an access control function which determines the most specific roles that do not contain any other extent of enabled roles. The computation of this not very efficient since the relation that defines the enabled roles in a location changes with each movement of a user or a change in spatial configuration of the environment. For dynamic environments like mobile networks the use of relational model for definition of spatial configuration is not very efficient. In the FPM-RBAC model the multi-domain mobile network and the associated security policies are formalized with Ambient Calculus and Ambient Logic which are dynamic and mobility-oriented formalisms.

5. XFPM-RBAC: XML BASED SPECIFICATION LANGUAGE FOR SECURITY POLICIES IN MULTI-DOMAIN MOBILE NETWORKS

XFPM-RBAC is a XML based security policy specification language based on the FPM-RBAC policy model. The structure of XFPM-RBAC is in the form of XML schemas and XSLT transformations. XML schemas present a formal way to express the mathematical model in plain text and exchange data structures with the applications utilizing XFPM-RBAC. Another advantage is support for the XSLT language, which is used in XFPM-RBAC to produce formal specifications from XML descriptions of multi-domain security policies.

In the following sections, we present the constructs of XFPM-RBAC. In Section 5.1, we present XML representation of data sets which make up the configuration of a domain. XML schemas for inter-domain role hierarchies and inter-domain role maps are explained in Section 5.2. The XML schema for security policies, presented in Section 5.3, is the basis for defining domain and inter-domain security policies. Location Formulas within the XML schema are XML representations of formal Ambient Logic formulas. The Location Formulas are presented in Section 5.4. In this chapter, we give the outlines of the XML schemas. The full specification of the XML schemas are presented in Appendix A.

The XML specifications of XFPM-RBAC policies are translated to formal specifications using transformations defined in XSLT language. In Section 5.5 we explore how the XSLT language is utilized for generating formal Ambient Calculus and Ambient Logic specifications from security policies. By the help of this translation, the security administrator does not need to be involved in the specifics of formal languages. The formal specifications are input to the Ambient Calculus Model Checker presented in Chapter 6, for the purpose of automated verification. Separation-of-duty (SOD) constraints are discussed in Section 5.6.

5.1. Domain Configurations in XFPM-RBAC

A *Domain* in XFPM-RBAC is a collection of sets containing set of Hosts, Users, Objects, as well as an Object Type Hierarchy, a Role Hierarchy, and attributes defining the name and the identifier (ID) of the domain. A separate administration interface for individual domains are used. Not all information belonging to domains is shared between multiple domains, due to the principle of distributed administration. The outline of XML schema of a Domain is presented in Figure 5.1. For sake of readability, the closing tags for schema elements and types are omitted from the listings and their order may be inferred from the tabular format.

In this XML schema, a *Domain* is defined by the following sequence: A set of *Hosts*, a set of *Users*, a *Role_Hierarchy*, an *Object Type Hierarchy* (*O_T_H*) and a set of *Objects*. A Domain has the attributes *Domain_ID*, which is the unique ID of the domain in the network, and *Domain_Name*, which is the name of the domain. The type *User_Def* defines a member of Users, *Host_Def* defines a member of Hosts, *Role_Def* defines a Role, *Object_Def* defines a member of Objects. The element *Object_Type* is the object type associated with a member of Objects.

An user definition *User_Def* includes a Name, Surname, an User ID, and the Home Domain ID of the user. The definition also includes a sequence of *Assigned_Role* elements which specify the set of roles that the User may be assigned. The *Assigned_Role* element is of XML complex type *Role_Assignment*. *Role_Assignment* includes the ID of the assigned role and a *Constraint*. A *Constraint* may include a location constraint and a generic constraint. This way, constraints may be applied to the user assignment relation.

The role hierarchies and object type hierarchies (*O_T_H*) in the Domain configurations are defined as follows. A *Role_Hierarchy* is a sequence of *Role* elements. A *Role* element which is of XML complex type *Role_Def* includes the ID of the parent role, a role ID and a role name. The *O_T_H* definition consists of a sequence of *O_T_H_Node* elements. Each *O_T_H_Node* elements includes a reference to a parent object type

```
<xs:schema... id="Domain">
  <xs:element name="Domain_Def">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="Hosts">
            <xs:element name="Users">
            <xs:element name="Role_Hierarchy">
            <xs:element name="O_T_H">
            <xs:element name="Objects">
         <xs:attribute name="Domain_ID" type="xs:ID"
            use="required">
         <xs:attribute name="Domain_Name" type="xs:string"
            use="required">
   <xs:complexType name="User_Def">
   <xs:complexType name="Host_Def">
   <xs:complexType name="Role_Def">
   <xs:complexType name="Constraint_Def">
   <xs:complexType name="Object_Def">
   <xs:complexType name="Object_Type_Def">
   <xs:element name="Role_Hierarchy">
   <xs:element name="O_T_H">
   <xs:complexType name="Role_Assignment">
   <xs:complexType name="O_T_H_Node">
   <xs:element name="Object_Type">
```

Figure 5.1. Outline of XML Schema of a Domain.

and an object type, both of which have XML complex types of *Object_Type_Def*.

5.2. Inter-Domain Configurations in XFPM-RBAC

Inter-domain configurations are used for exporting role hierarchies and mappings to other domains as well as defining role maps for the home domain. Inter-domain configurations in XFPM-RBAC consists of the following information: The home domain role hierarchy RH_h, a foreign domain role hierarchy RH_f, an inter-domain role hierarchy RH_Γ, a home role map RM_h and a foreign role map RM_f. RH_h is defined by the home administrator in the user interface (SPMI) of the home domain. This information is not exported to other domains, therefore it may be empty in an exported XML file for inter-domain configuration. RH_f is the role hierarchy for a foreign domain. RH_f is defined by a foreign domain administrator in the administration interface of a foreign domain. It is imported to the home domain administration interface with the inter-domain configuration. RH_Γ defines the inter-domain roles and their inheritance relations.

A role map is a one-to-many relationship that associates an inter-domain role to a set of home or foreign roles. A foreign role map RM_f provides a mapping from foreign roles to the inter-domain roles. This information may be defined by home or foreign administrators. A home role map RM_h defines the mapping from home domain roles to inter-domain roles and is defined by the home domain administrator.

The outline of XML schema for inter-domain configurations is presented in Figure 5.2. There are some identity constraints for inter-domain configurations to ensure consistency among multiple role hierarchy definitions. The *Role_ID_unique* constraint ensures unique role IDs (IDs) among multiple domains. The *Home_Role_ID_key* constraint defines the key for RH_h. *Home_Role_Map_keyref* constraint restricts the mapped role ID values of RM_h to the role IDs for RH_h. *Home_Role_ID_keyref* restricts the parent role ID values of RH_h to the role IDs in RH_h. The *Interdomain_Role_ID_keyref* constraint restricts the interdomain role IDs to the values of the role IDs for RH_Γ. Finally, *ForeignRoleID_keyref* restricts the mapped role ID

```
<xs:element name="Interdomain_Def">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="Role_Hierarchy" minOccurs="0">
            <xs:element name="Foreign_Role_Hierarchy">
            <xs:element name="Interdomain_Role_Hierarchy">
            <xs:element name="Role_Map_Home">
            <xs:element name="Role_Map_Foreign">
   <xs:keyref name="Interdomain_Role_ID_keyref"
      refer="Interdomain_Role_ID_key">
   <xs:keyref name="ForeignRoleID_keyref"
      refer="ForeignRoleID_key">
   <xs:unique name="Role_ID_unique">
   <xs:key name="Home_Role_ID_key">
   <xs:keyref name="Home_Role_Map_keyref"
      refer="Home_Role_ID_key">
   <xs:keyref name="Home_Role_ID_keyref"
      refer="Home_Role_ID_key">
```

Figure 5.2. Outline of XML Schema for Inter-Domain Configurations.

values for RM_f to the role IDs of RH_f.

RH_Γ defines the inheritance relations between inter-domain roles. An inter-domain role is a role for inter-domain access. It is defined by the structure *Role_Def* which is the same for other type of roles. Identity constraints for inter-domain role definitions are as follows: *Interdomain_Role_ID_key* constraints defines *Role_ID* as a key for RH_Γ and *Interdomain_keyref* constraint restricts *Parent_Role* field to values of *Interdomain_Role_ID_key* in RH_Γ. The XML Schema for inter-domain role hierarchies is given in Figure 5.3.

A home or foreign role map associates one inter-domain role to a set of home or foreign roles. *Role_Map_Home* is the home role map RM_h and *Role_Map_Foreign* is the foreign role map RM_f. Home and foreign role maps are defined by a sequence

```
<xs:element name="Interdomain_Role_Hierarchy">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="Interdomain_Role" type="Role_Def"
               maxOccurs="unbounded">
      <xs:key name="Interdomain_Role_ID_key">
         <xs:selector xpath="Interdomain_Role">
         <xs:field xpath="Role_ID">
      <xs:keyref name="Interdomain_keyref"
         refer="Interdomain_Role_ID_key">
         <xs:selector xpath="Interdomain_Role">
         <xs:field xpath="Parent_Role">
```

Figure 5.3. XML Schema for Inter-Domain Role Hierarchies.

of *Role_Map* elements, which are of XML complex type *Role_MapT*. Each *Role_Map* element includes an inter-domain role defined by *Interdomain_Role_ID* and a set of mapped roles (*Mapped_Role*). Mapped roles are defined by role IDs of home or foreign role hierarchies, depending on whether a home or foreign role map is specified. The XML Schema for role maps is provided in Figure 5.4.

```
   <xs:complexType name="Role_MapT">
      <xs:sequence>
         <xs:element name="Interdomain_Role_ID" type="xs:IDREF">
         <xs:sequence>
            <xs:element name="Mapped_Role"
               maxOccurs="unbounded">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="Mapped_Role_ID"
                        type="xs:IDREF">
   <xs:element name="Role_Map" type="Role_MapT">
```

Figure 5.4. XML Schema for Role Maps.

5.3. Multi-Domain Security Policies in XFPM-RBAC

Security policy definitions in XFPM-RBAC are multi-domain, i.e. they define both domain and inter-domain security policies. A security policy definition consists of services, policy rules and location constraints. Services define the subset of authorization objects which may be included in the policy rules. The policy rules define the authorization terms which include permissions associated with location, mobility and generic constraints. Both the permission assignment matrix (PAM) and the authorization terms specified by the Security Administrator in the SPMI are represented with policy rules. The outline of XML schema of a security policy is presented in Figure 5.5.

The *Services* element consists of a sequence of elements of type *Service*. The *ServiceIDUnique* constraint enforces the $Service\_ID$ to be unique within the security policy. $Policy\_Rules$ consists of a (possibly empty) sequence of $Policy\_Rule$ elements. The $is\_interdomain$ attribute of the $Policy\_Rules$ element indicates whether the rule is an inter-domain security policy rule. Location and mobility constraints are formalized using Location Formulas. The $Location\_Formulas$ element is made up of a (possibly empty) sequence of elements of type $Location\_Formula$. The location and mobility constraints are presented in Section 5.4.

A *Service* element in XFPM-RBAC includes an attribute $is\_interdomain$ which indicates whether the service is an inter-domain service. $Service\_Name$ specifies the name of the service and $Service\_ID$ gives a unique ID to the service. A *Service* includes a possibly empty set of $Service\_Host$ elements. $Service\_Host$ specifies a host (of type Host_Def) associated with the service, in addition to a group of actions (of type $Service\_Member\_Action\_Group$) which may be executed on the host. The attribute srv_m_id gives an identity to membership of a service. In a similar fashion, $Service\_Domain$, $Service\_Object\_Type$ and $Service\_Object$ elements represent a set of domains, object types and objects as well as a set of actions associated with the service. For domain services, the service is associated only with the home domain, whereas inter-domain services may be associated with multiple domains. The *Mem-*

```
<xs:element name="Security_Policy">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="Services">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element ref="Service"
                        maxOccurs="unbounded">
               <xs:unique name="ServiceIDUnique">
                  <xs:selector xpath="Service">
                  <xs:field xpath="Service_ID">
            <xs:element name="Policy_Rules">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element ref="Policy_Rule"
                        minOccurs="0" maxOccurs="unbounded">
                  <xs:attribute name="is_interdomain"
                     type="xs:boolean">
            <xs:element name="Location_Formulas">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element ref="Location_Formula"
                        minOccurs="0" maxOccurs="unbounded">
```

Figure 5.5. Outline of XML Schema of a Security Policy.

berIDUnique constraint ensures that each association of an authorization to a service is given a unique identifier.

A *Policy_Rule* element in XFPM-RBAC includes attributes *auth_obj_type* and *service_member_ref*, which are the object type and service associated with the authorization object referred in the policy rule, and the attribute *role_ref*, which is the identifier of the assigned role of the authorization subject. The 5-tuple authorization term is represented with the sequence of elements consisting of *Authorization_Subject*, *Authorization_Object*, *Rule_Action*, *Location_Formula_Name* and *Conditions*. The first three elements in the sequence are mandatory, whereas the final two are optional, since some rules may not be associated with constraints.

5.4. Representation of Location and Mobility in XFPM-RBAC

The location and mobility model in XFPM-RBAC is utilized for two purposes: First, a formal location and mobility model of the network, called a *location configuration*, models the state of the network. Second, the location and mobility constraints in the security policy rules represented as *location formulas* model spatio-temporal constraints to be satisfied by the network. Location configuration is represented with Ambient Calculus and location formulas are represented with Ambient Logic.

In this model, locations are represented with Ambients. Location hierarchies are represented with ambients which are contained within each other. The XML representations of Ambient Calculus and Ambient Logic are based on syntax defined in BNF notation. The BNF notations have been developed using the ANTLR lexer-parser tool.

The BNF Grammar for Ambient Calculus Syntax definition in XFPM-RBAC is presented in Figure 5.6. An Ambient Calculus specification consists of a *Composition* term, which is a combination of *Sequence* terms separated with the *Parallel* (|) operator. The parallel operator defines locations in the same level of the location configuration. For example, the specification $Host1[File1[]|File2[]]$ defines a host with

```
CompilationUnit   ::=   Specification ";"
Specification  ::=   <ID> ":::=" Composition
Composition ::=   Sequence ( "|" Sequence )*
Sequence ::=   Path BasicExpresion
BasicExpresion ::=   "0" | <ID> "[" Composition "]" | <ID> "["
   "]" | "{" Composition "}"
Path  ::=   ( Action "." )*
Action   ::=   ( "in" <ID>|"out" <ID> | "mv_in" <ID> | "mv_out"
   <ID> | "acid" <ID> | "open" <ID> | "(" <ID> ")" | "<" <ID>
   ">")
```

Figure 5.6. BNF Grammar for Ambient Calculus Syntax in XFPM-RBAC.

two files. A *Sequence* consists of a *Path* followed by a *BasicExpression*. A *Path* is a sequence of Ambient Calculus actions (capabilities) which are separated with ".".

The BNF Grammar for Ambient Logic Syntax definition in XFPM-RBAC is presented in Figure 5.7. Here, a *Formula* is made up of logical sentences (*LogicalS*) connected with the logical *Or* operator ($\vee$). Each of these may be a set of *Unary* terms connected with the *Parallel* ($|$) operator. Parallel operator defines two locations in parallel, which are in the same level in location configuration. For example the term $Student[]|PC\_Lab\_Client[]$ states that Student and PC_Lab_Client are in the same location. A *Unary* term may consist of a *BasicFormula* or another Unary term which may be associated with a *Temporal*, *Spatial* or *Negation* operator. A Temporal operator defines temporal logic operators of "**AG**" (Always) and "**EF**" (Eventually). These operators are defined in CTL temporal logic. A Spatial operator defines Ambient Logic spatial operators of "$\diamond$" (Somewhere) and "**EW**" (Everywhere). A *BasicFormula* may include the *Inactivity* operator (0) which represents a location which does no action, the *True* operator which represents any location, an empty location (e.g. $File[]$) or a location definition which includes other logical sentences (e.g. $Server[User[T]]$).

An example location formula defined with the grammar presented above is as follows: The formula "$f ::= AG - SW\{file2[SW\{data1[T]|T\}]|T\};$" states that, it

```
CompilationUnit   ::=   Formula ";"
Formula  ::=   <ID> ":=" LogicalS
LogicalS ::=   Or ( "==" Or )*
Or ::=   Composition ( "+" Composition )*
Composition ::=   Unary ( ( "|" ) Unary )*
Unary ::=   BasicFormula |
   ((Negation|TemporalUnary|SpatialUnary)Unary)
Negation ::=   ( "-" )
TemporalUnary  ::=   ( ( "AG"|"EF" ) )
SpatialUnary   ::=   ( "SW"|"EW" )
BasicFormula   ::=   "0" | "T" | <ID> "[" LogicalS "]" |
   <ID>"[""]"
```

Figure 5.7. BNF Grammar for Ambient Logic Syntax in XFPM-RBAC.

should never be valid that, somewhere in the location configuration, there is a location called *file2* which contains *data1* somewhere within itself. It may contain anything else, and there may be anything else together with *file2* in the same location. This formula would match to a location configuration of the network (LCONF) which contains *file2* which contains *data1*. The negation in the beginning of the formula is necessary for model checking since model checker only finds counter-examples (as opposed to proofs in theorem provers).

The XML schemas for the Ambient Calculus and Ambient Logic specifications reflect the structure of BNF grammar. Because these specifications are lengthy, we do not provide the details. An Ambient Calculus specification consists of a formal specification referred as *Compilation_Unit*, which is the top element corresponding to the BNF grammar, and a section *Variables*, which is a mapping from names of locations in the network to identifiers used in the formal specification. The variable identifiers are defined with XML keys and referred with XML keyref definitions.

A location formula is in the form of "$Formula\_Name ::= Logical\_Expr$;", where *Logical_Expr* is an XSD element corresponding to an Ambient Logic logical sentence

```
<xs:element name="Ambient_Calculus_Spec">
   <xs:complexType>
      <xs:sequence>
         <xs:element ref="Compilation_Unit">
         <xs:element name="Variables">
      <xs:attribute name="Ambient_Calculus_Spec"
         type="xs:string" use="required">
   <xs:keyref name="Id_Ref_Amb" refer="Id_Key">
   <xs:keyref name="Id_Ref" refer="Id_Key">
   <xs:key name="Id_Key">
```

Figure 5.8. Outline of XML Schema of Ambient Calculus Specifications.

LogicalS referred in Figure 5.7. The structure of the XSD schemas presented in Figure 5.8 and Figure 5.9 reflect the BNF notations of Ambient Calculus and Ambient Logic.

A XML specification for an example formula $spec1 ::= AG-SW\{file2[]\};$, which states that "it should never become true that: an empty file called file2 is somewhere

```
<xs:element name="Location_Formula">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="Formal_Expr" type="xs:string"
            minOccurs="0"/>
         <xs:element name="Formula_Name" type="xs:ID"/>
         <xs:element name="Congruence" fixed="::=">
            <xs:simpleType>
               <xs:restriction base="Location_Tokens">
                  <xs:enumeration value="::="/>
         <xs:element ref="Logical_Expr"/>
         <xs:element name="Formula_End"
            type="Location_Tokens" fixed=";"/>
```

Figure 5.9. Outline of XML Schema of Location Formulas.

```
<Congruence>::=</Congruence>
<Logical_Expr><Or><Composition><Everytime token ="AG">
<Not_Expr token="-"><Somewhere token = "SW"><Basic_Formula_Expr>
<Bracket><Logical_Expr><Or><Composition><Basic_Formula_Expr>
<Ambient><AmbientName>file2</AmbientName></Ambient>
</Basic_Formula_Expr></Composition></Or></Logical_Expr>
</Bracket></Basic_Formula_Expr></Somewhere>
</Not_Expr></Everytime></Composition></Or>
</Logical_Expr><Formula_End>;</Formula_End>
```

Figure 5.10. An example XML specification for a Location Formula.

in the network", is given in Figure 5.10. The negation phrase in the beginning is necessary for finding counter-examples in the model checker. The second part of the formula specification is the phrase formalizing the location and mobility constraint.

Although a XML specification for a location and mobility constraint is lengthy and complex, the user is not concerned with this issue since these specifications are defined and generated via the Security Policy Management Interface.

5.5. Generation of Formal Specifications from Security Policy using XSLT

The XFPM-RBAC policy specification language makes use of the XSLT language. XSLT is a language for transforming the structure and content of a XML document. In the context of security policies, XFPM-RBAC uses XSLT for generation of formal specifications in Ambient Logic and Ambient Calculus from security policy specifications and domain configurations in XML format. The generation of formal specifications takes place in two steps. First, location and mobility constraints in the security policy rules are converted to formal Ambient Logic formulas. We present this step in Section 5.5.1. Second, domain configurations and allowed actions within the security policy specifications are converted into Ambient Calculus process specifications. We present this step in Section 5.5.2. The generated files are used as output to the Ambient Calculus Model Checker presented in [41], for the purpose of checking satis-

faction of Ambient Logic formulas against Ambient Calculus process specifications.

5.5.1. Translation of Location Formulas in Security Policy Rules to Formal Specifications using XSLT

The XSLT transformation for translation of location formula specifications in XML to formal Ambient Logic specifications takes place as follows: For each construct in the language, there is a XML element that defines the construct in the formal language. These constructs are based on the BNF notation of Ambient Logic reflected to a XML schema structure. In order to translate from a XML specification to a formal specification, XSL templates are utilized. There is a XSL template for each construct in the XML schema that will be translated to formal language. As an example, the XSL template for the logical expressions, namely the template for *Logical_Expr* element, is shown in Figure 5.11. A logical expression may include an *Or* expression which consists of *Composition* expressions combined with +. Each *Composition* sub-expression may consist of further sub-expressions combined with the parallel operator |. These sub-expressions may be in the form of a *Basic Formula*, may contain temporal operators *Everytime* (AG) and *Sometime* (EF), or may contain spatial operators *Somewhere* ($\diamond$) and *Everywhere* (EW), or may contain a negation operator ($\neg$). Each of these sub-expressions are handled with a separate template match operation in XSLT. Because of lack of space we include an outline of one of the templates in this paper.

When the XSLT transformation for Location Formulas is applied to a Security Policy, a set of formal specifications are generated for each location formula in the security policy rules. For the example in Figure 5.10, the application of XSLT results with the formula $spec1 ::= AG - SW\{file2[]\};$. The location formulas are defined through the SPMI application therefore the formal specifications for locations formulas are defined in an automated fashion without the need for formal methods knowledge of the security administrator.

```
<xsl:template match="Logical_Expr">
   <xsl:for-each select="Or">
         <xsl:if test= "exists(preceding-sibling::element())">
               <xsl:if test= "parent::Logical_Expr/@token">
                  <xsl:value-of
                     select="parent::Logical_Expr/@token">
                  </xsl:value-of>
               </xsl:if>
            </xsl:if>
         <xsl:for-each select="Composition">
            <xsl:if test="exists␣
               ((preceding-sibling::element()))">
               <xsl:if test="parent::Or/@token">
                  <xsl:value-of
                     select="parent::Or/@token"></xsl:value-of>
               </xsl:if>
            </xsl:if>
            <xsl:apply-templates select="Basic_Formula_Expr"/>
            <xsl:apply-templates select="Everytime"/>
            <xsl:apply-templates select="Everywhere"/>
            <xsl:apply-templates select="Sometime"/>
            <xsl:apply-templates select="Somewhere"/>
            <xsl:apply-templates select="Not_Expr"/>
      </xsl:for-each>
   </xsl:for-each>
</xsl:template>
```

Figure 5.11. Outline of XSL Template for a Logical Expression in a Location Formula.

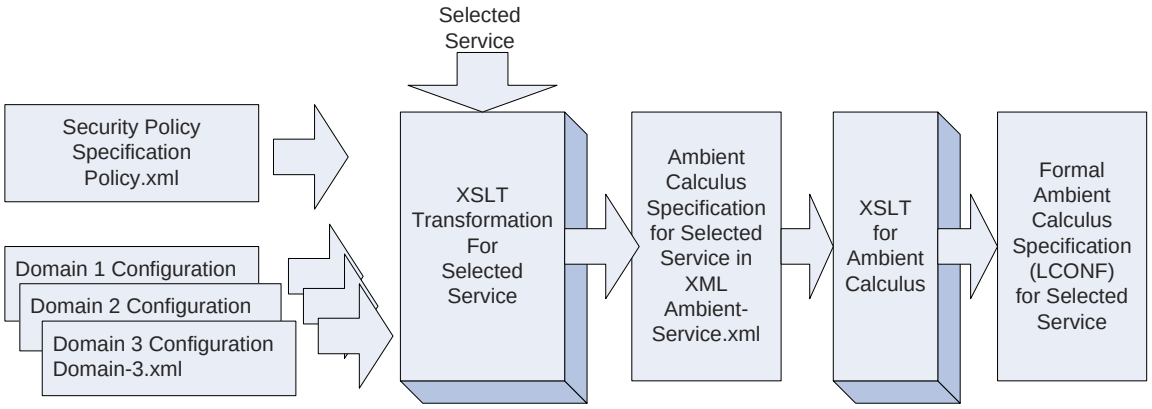

Figure 5.12. Generation of formal location configuration specifications using XSLT.

5.5.2. Generation of Location Configurations from Multiple Security Policy Definitions using XSLT

Before generation of formal specifications, the security policies and the domain configurations are specified according to XFPM-RBAC schemas. A domain configuration consists of the data sets and relations specified for a certain domain as presented in Section 4.1. 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 Location Configuration (LCONF).

The Location Configuration (LCONF) for a network reflects the current state of the security policy in terms of location hierarchies and actions allowed by entities in the security policy. LCONF is a formal specification in Ambient Calculus. In order to generate LCONF, information from multiple XML files are used. A service is selected for generating formal specifications pertaining to a specific service. The domain configuration, inter domain configuration as well as security policy definitions are brewed together and transformed using XSLT to generate a XML specification compliant to the XSD for Ambient Calculus presented in Figure 5.8. A second XSLT transformation is applied to the XML specification for generation of LCONF in formal language, based on the BNF notation presented in Figure 5.6. This process is summarized in Figure 5.12.

Due to the length and complexity of the XSLT transformation to generate XML specifications of the location configuration for a service, we hereby present the pseudocode of the transformation in Figure 5.13. In this transformation, each Service corresponds to a different Ambient Calculus Specification. Each *Domain* definition in the domain configuration files corresponds to an *Ambient Expression*. Within the domain ambient, a *Host* and a *User* is added as ambients if they are enrolled in the domain. Within *Host* ambients, *Object* ambients are added if they are registered with the host. The *Service* definition contains member elements for *Service_Domain*, *Service_Host* and *Service_Object* corresponding to domains, hosts and objects associated with the service. These elements are used to filter domains, hosts and objects to be included in the formal specification. A separate section for *Variables* is added to the XML specification. The Variables section contains *Identifier* elements for ambients to provide a mapping of authorization object identifiers to their names. This section is used by the SPMI and the client applications to map the identifiers in the formal specifications to the names in domain configuration, inter domain configuration and security policy definitions.

5.6. Separation of Duty (SOD) Constraints in XFPM-RBAC

XFPM-RBAC includes XML based constructs for the specification of separation of duty constraints based on the FPM-RBAC model. A SOD constraint is specified by the abstract element *SOD_Constraint* which is of type *SOD_Constraint_Def*. An abstract element does not take place in a XML document, it defines a class which needs to be instantiated with another element. Each type of SOD constraints inherit *SOD_Constraint*, and, if necessary, extend *SOD_Constraint* with additional constraints. A SOD constraint is identified by *constraint_id*. Depending on the type of SOD constraint, it includes a reference to a conflicting set of roles or services with (*conflicting_set_id*). The parameter *n_conflicting* specifies the maximum number of elements from a conflicting set of roles or services, that may be assigned to an user or role. The attribute *is_dynamic* is specified if the SOD constraint is a dynamic SOD constraint. The specifications are outlined in Figure 5.14.

```
for-each Service select
   where Service_Name = sname
      for each Service/Service_Domain
         for-each Domain_Def select
            where Service_Domain/srv_m_id=Domain_Def/Domain_ID
            Add an Identifier for the Domain
            Add an Ambient_Expr for the Domain
            for-each Domain_Def/Host select where
               Enrolled_Domain_ID = Domain_ID
               for-each Service/Service_Host where
                  srv_m_id=Host_ID
                  Add an Identifier for the Host
                  Add an Ambient_Expr for the Host
                  Compose Host's with "|" operator
            for-each Domain_Def/Object select where
               Host/Host_Object_ID = Object_ID
               for-each Service/Service_Object where
                  srv_m_id=Object_ID
                  Add an Identifier for the Object
                  Add an Ambient_Expr for the Object
                  Compose Object's with "|" operator
      for-each Domain_Def/User select where Home_Domain_ID =
         Domain_ID
         for-each Service/Service_Host where srv_m_id=User_ID
                  Add an Identifier for the User
                  Add an Ambient_Expr for the User
                  Compose User's with "|" operator
            Compose Domain's with "|" operator
```

Figure 5.13. Pseudocode for XSLT to generate XML specification of the location configuration for a service.

```
<xs:element name="SOD_Constraint" type="SOD_Constraint_Def"
   abstract="1"/>
<xs:complexType name="SOD_Constraint_Def">
   <xs:attribute name="constraint_id" type="xs:ID"
      use="required"/>
   <xs:attribute name="conflicting_set_id" type="xs:IDREF"
      use="required"/>
   <xs:attribute name="n_conflicting" type="xs:integer"
      use="optional"/>
   <xs:attribute name="is_dynamic" type="xs:boolean"
      use="optional"/>
</xs:complexType>
```

Figure 5.14. Definition of abstract element for SOD constraints.

Role-based SOD (SOD_CR) is defined with respect to a set of conflicting roles *CR*. According to SOD_CR, n or more members of the role set CR may not be assigned to a single user. SOD_CR is a constraint on the user assignment (UA) relation. SOD_CR is an instance of the abstract element *SOD_Constraint*. The definition of a conflicting role set *CR* is given in Figure 5.15. A *Conflicting_Role_Set* includes a sequence of *Conflicting_Role*'s, a set identifier (*set_id*) and *num_roles*, which is the number of conflicting roles within the conflicting role set. A *Conflicting_Role* is identified by the ID of the role which is designated as a conflicting role (*cr_id*).

Service-based SOD (SOD_CS) is defined with respect to a set of conflicting services *CS*. According to SOD_CS, n or more of the members of *CS* may not be assigned to a single role. SOD_CS is a constraint on the service access matrix (SAM). SOD_CS is also an instance of the abstract element *SOD_Constraint*. The definition of a conflicting service set *CS* is given in Figure 5.16. A *Conflicting_Service_Set* includes a sequence of *Conflicting_Service*'s, a set identifier (*set_id*) and *num_services*, which is the number of conflicting services within the conflicting service set. A *Conflicting_Service* is identified by the ID of the service which is designated as a conflicting service (*cs_id*).

```
<xs:element name="SOD_CR" type="SOD_Constraint_Def"
   substitutionGroup="SOD_Constraint"/>
<xs:element name="Conflicting_Role_Set" minOccurs="0"
   maxOccurs="unbounded">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="Conflicting_Role"
            maxOccurs="unbounded">
            <xs:complexType>
               <xs:attribute name="cr_id" type="xs:string"
                  use="required"/>
         <xs:attribute name="set_id" type="xs:ID"
            use="required"/>
         <xs:attribute name="num_roles" type="xs:integer"
            use="optional"/>
```

Figure 5.15. Definition of role based SOD.

```
<xs:element name="SOD_CS" type="SOD_Constraint_Def"
   substitutionGroup="SOD_Constraint"/>
<xs:element name="Conflicting_Service_Set" minOccurs="0"
   maxOccurs="unbounded">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="Conflicting_Service"
            maxOccurs="unbounded">
            <xs:complexType>
            <xs:attribute name="cs_id" type="xs:string"
               use="required"/>
      <xs:attribute name="set_id" type="xs:ID" use="required"/>
      <xs:attribute name="num_services"/>
```

Figure 5.16. Definition of service based SOD.

```
<xs:element name="SOD_ICR"
substitutionGroup="SOD_Constraint">
   <xs:complexType>
      <xs:complexContent>
         <xs:extension base="SOD_Constraint_Def">
            <xs:attribute name="mapped_conflicting_set"
               type="xs:IDREF" use="required"/>
            <xs:attribute name="home_foreign" use="optional"
               default="HOME">
               <xs:simpleType>
                  <xs:restriction base="xs:string">
                     <xs:enumeration value="HOME"/>
                     <xs:enumeration value="FOREIGN"/>
```

Figure 5.17. Definition of inter-domain SOD.

Inter-domain SOD (SOD_ICR) is related with inter-domain role mapping. According to SOD_ICR, no user can be assigned to a set of roles which map to n or more conflicting inter-domain roles. SOD_ICR is a constraint on the role map (RM) relation. The definition of SOD_ICR is an extension of the abstract element *SOD_Constraint*. The SOD_ICR element which is presented in Figure 5.17 includes additional attributes of *mapped_conflicting_set* and *home_foreign*. The attribute *mapped_conflicting_set* specifies the home or foreign conflicting role set for mapping to assigned roles. The *home_foreign* attribute may take two values: HOME and FOREIGN based on whether the role map relation RM is a home role map or a foreign role map.

Location based SOD constraint (SOD_LCR) introduces a location and mobility aspect to the SOD constraint. Two or more roles or permissions may be considered in conflict if one of these permissions relates to an action which involves locations and mobility. According to the location based SOD constraint (SOD_LCR), n or more roles may not be assigned to an user if these roles are within the conflicting roles set CR and if the Location constraint *sfo* is valid in the initial location configuration of

```
<xs:element name="SOD_LCR"
substitutionGroup="SOD_Constraint">
   <xs:complexType>
      <xs:complexContent>
         <xs:extension base="SOD_Constraint_Def">
            <xs:attribute name="sfo" type="xs:IDREF"
               use="required"/>
```

Figure 5.18. Definition of location based SOD.

the system, i.e. $LCONF_0 \vDash sfo$. The *SOD_LCR* element which is presented in Figure 5.18 is also an extension of *SOD_Constraint*. The additional attribute is a location constraint *sfo* that is identified by the reference ID of a previously specified location formula.

Dynamic SOD constraints are based on activation of roles, which occurs when an user logs into a service, rather than assignment or roles, and they are evaluated at run-time against the current state of the system, in which a set of actions has already been executed. Dynamic SOD constraints have the same types as their static counterparts and differ only on the time of evaluation. The evaluation of constraints takes place in the enforcement logic of application that implements XFPM-RBAC. For this reason, XFPM-RBAC does not provide separate definitions for dynamic SOD constraints; their specification is the same as static SOD constraints. The distinction is the specification of *is_dynamic* attribute for dynamic constraints.

XFPM-RBAC includes support for evaluation of static SOD constraints for applications. This takes place through XSLT transformations which evaluate conflicts for a given set of SOD constraint definitions, user and service assignments in the multi-domain security policies. We give the result of evaluation of a SOD constraint as part of the case study in Section 8.2.5.

6. MODEL CHECKING OF SECURITY POLICIES WITH AMBIENT CALCULUS

The main idea in model checking is to define an automated mechanism which explores all possible states of a system and test these states with respect to a set of desired properties. The set of the possible states of a system must be finite for such an exploration to be performed. Model checking can be divided into three processes as 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 the 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.

The details of the approach presented in this chapter may be found in [67]. In Section 6.1, we cover the concept of model checking for security policies. In Section 6.2, we provide an overview of formal semantics for Ambient Calculus and Ambient Logic. In Section 6.3, a model checking algorithm is proposed for model checking of systems modeled as Ambient Calculus specifications with respect to properties specified as Ambient Logic formulas. The model checking algorithm is the result of joint work, as part of M.S. thesis of Ozan Akar [1].

6.1. 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 location and mobility constraints in the security policy rules 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 location and mobility constraints in the security policy rules are satisfied by the network state.

6.1.1. 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 6.1, 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.

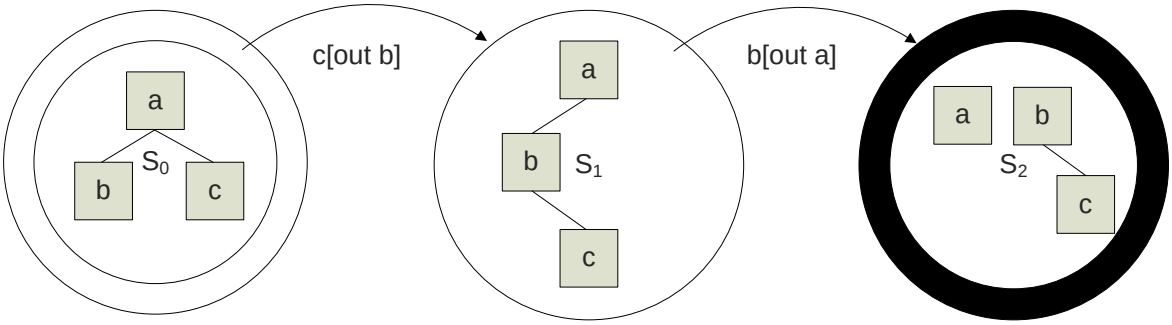

Figure 6.1. Events as a means to change the state of process specification.

An Ambient Calculus specification such as $d[inc.(R).P]$ is represented as a sequence of events called a Trace: $inc, (R), P$. 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 Security Policy Rule can be represented as a set of Preconditions and Postconditions. For example, for the rule "Project files in $Server_c$ can not 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.

6.1.2. 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. Model checker can generate a counter-example T that satisfies $\{P\}T\{Q\}$. This will result in a trace that represents a match of a location and mobility constraint in a security policy rule.

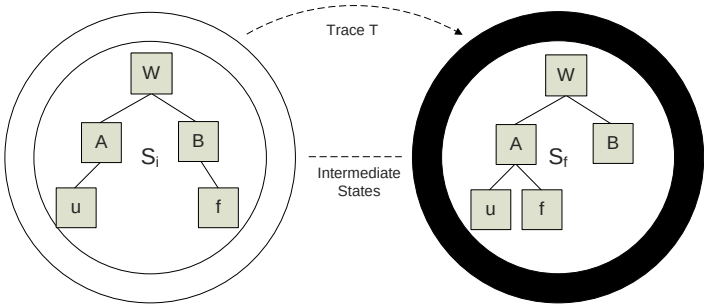

Figure 6.2. Finding a trace T that leads to a final state from an initial state.

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 6.2 can be such that "Files in Domain B can not 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 which can be accomplished by model checking.

Ambient Calculus model checking can be decomposed into two major search problems. The first problem is to search future evolutions of a given model. A logic formula can include constructs quantify rest of formula over future states. When evaluating truth of a formula with respect to an Ambient Calculus specification, analysis of reachable future states is needed. The second problem is to search spatial congruence of model and the logic formula. Both Ambient Calculus specification 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.

Table 6.1. Structural congruence for Ambient Calculus specifications.

Structural Congruence Rule	**Structural Congruence Rule**
$P \equiv P$	$P \equiv Q \Rightarrow Q \equiv P$
$P \equiv Q, Q \equiv R \Rightarrow P \equiv R$	$P \equiv Q \Rightarrow (\nu n)P \equiv (\nu n)Q$
$P \equiv Q \Rightarrow P\|R \equiv Q\|R$	$P \equiv Q \Rightarrow n[P] \equiv n[Q]$
$P \equiv Q \Rightarrow M.P \equiv M.Q$	$P \equiv Q \Rightarrow (n).P \equiv (n).Q$
$\epsilon.P \equiv P$	$(M.M').P \equiv M.M'.P$
$(\nu n)(\nu m)P \equiv (\nu m)(\nu n)P$	$(\nu n)\mathbf{0} \equiv \mathbf{0}$
$(\nu n)(P\|Q) \equiv P\|(\nu n)Q$ if $n \notin fn(P)$	$(\nu n)m[P] \equiv m[(\nu n)P]$ if $n \neq m$
$P\|\mathbf{0} \equiv P$	$P\|Q \equiv Q\|P$
$P\|Q \equiv Q\|P$	$(P\|Q)\|R \equiv P\|(Q\|R)$

6.2. Formal Semantics for Ambient Calculus and Ambient Logic

The formal semantics of Ambient Calculus specifications and Ambient Logic formulas constitute the basis of the presented model checking algorithm. The semantics are defined in [4] and [5]. The formal semantics of Ambient Calculus specifications are outlined in Section 6.2.1. The formal semantics of Ambient Logic formulas are outlined in Section 6.2.2.

6.2.1. Formal Semantics of Ambient Calculus Specifications

Structural congurence relation preserves equivalence of processes up to trivial syntactic restructuring. Structural congruence is the basis for definition of reduction relation. The reduction relation, describes the dynamic behavior of ambients. A reduction relation $P \rightarrow Q$ describes the evolution of a process P into a new process Q. Structural congruence and reduction relations of Ambient Calculus are defined in [5]. In Table 6.2.1 we provide definitions of structural congruence and in Table 6.2.1 we provide definitions of reduction relation. Since we use a replication-free fragment of Ambient Calculus in the thesis in order to achieve a decidable model checking algorithm, the definitions related to replication are excluded from these definitions.

Table 6.2. Reduction relation for Ambient Calculus specifications.

Reduction Rule	**Name of Reduction Rule**
$n[in\ m.P\|Q]\|m[R] \rightarrow m[n[P\|Q]\|R]$	Red In
$m[n[out\ m.P\|Q]\|R] \rightarrow n[P\|Q]\|m[R]$	Red Out
$open\ n.P\|n[Q] \rightarrow P\|Q$	Red Open
$P \rightarrow Q \Rightarrow (\nu n)P \rightarrow (\nu n)Q$	Red Res
$P \rightarrow Q \Rightarrow n[P] \rightarrow n[Q]$	Red Amb
$P \rightarrow Q \Rightarrow P\|R \rightarrow Q\|R$	Red Par
$P' \equiv Q, P \rightarrow Q, Q \equiv Q' \Rightarrow P' \rightarrow Q'$	Red $\equiv$
$\rightarrow^*$	Reflexive and transitive closure of $\rightarrow$

6.2.2. Formal Semantics of Ambient Logic Formulas

Modal logic is used for expressing properties of models which cannot be expressed by the constructs of calculi. The Ambient Logic is a modal logic for expressing spatial and temporal properties of Ambient Calculus. Ambient Logic is strictly based on Ambient Calculus; all the spatial and temporal constructs are reflected in the logic. The main differences of Ambient Logic from latter logics are more expressive space modalities and simpler temporal connectives.

The satisfaction relation $P \models \mathcal{A}$ means that the process P satisfies the closed formula $\mathcal{A}$. In Definition 6.1 we give the formal semantic definitions for Ambient Logic formulas defined in [4]. Here, Π is the sort of processes, Φ is the sort of formulas, ϑ is the sort of variables, and Λ is the sort of names.

Definition 6.1. *The satisfaction relation for Ambient Logic formulas are defined as follows.*

(i) The atomic formula T of Ambient Logic is satisfied by all processes of Ambient Calculus. $\forall P \in \Pi.P \models T$

(ii) Negation of a formula is satisfied by any process which does not satisfy original formula. $\forall P \in \Pi, \mathscr{A} \in \Phi.P \models \neg\mathscr{A} \stackrel{\triangle}{=} \neg P \models \mathscr{A}$

(iii) A process satisfies the formula $\mathscr{A} \vee \mathscr{B}$ *if it satisfies either* $\mathscr{A}$ *or* $\mathscr{B}$. $\forall P \in$

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

(iv) *The formula* 0 *is satisfied by only processes structurally congruent to inactivity process.* $\forall P \in \Pi . P \models 0 \stackrel{\triangle}{=} P \equiv 0$

(v) *The formula* $n[\mathscr{A}]$ *is satisfied by processes which are structurally congruent to* $n[P']$ *for any P' where* $\mathscr{A}$ *is satisfied by* P'. $\forall P \in \Pi, n \in \Lambda, \mathscr{A} \in \Phi . P \models n[\mathscr{A}] \stackrel{\triangle}{=} \exists P' \in \Pi .\ P' \models \mathscr{A} \wedge P \equiv n[P']$

(vi) *The formula* $\mathscr{A}|\mathscr{B}$ *is satisfied by any process that can be decomposable into two processes as* $P'|P''$ *where* P' *satisfies* $\mathscr{A}$ *and* P'' *satisfies* $\mathscr{B}$. $\forall P \in \Pi, \mathscr{A}, \mathscr{B} \in \Phi . P \models \mathscr{A}|\mathscr{B} \stackrel{\triangle}{=} \exists P', P'' \in \Pi .\ P \equiv P'|P'' \wedge P' \models \mathscr{A} \wedge P'' \models \mathscr{B}$

(vii) *The nesting relation, denoted by* $\downarrow$, *is defined over two processes as* $P \downarrow Q$ *and indicates that Q is nested one level down in any ambient which exists at the top of the topology of P.* $P \downarrow P' \quad \text{iff } \exists\, n, P'' . P \equiv n[P']|P''$

(viii) *Relation* $\downarrow^*$ *is reflexive transitive closure of* $\downarrow$. $P \downarrow^* Q$ *indicates that P contains Q in somewhere of its topology.* $\downarrow^*$ is the reflexive and transitive closure of $\downarrow$

(ix) *Somewhere connective,* $\diamond$, *is used for specifying nesting properties of processes on the basis provided by the nesting relation defined above. The formula* $\diamond\mathscr{A}$ *is satisfied by processes which satisfies A in some inner location.* $\forall P \in \Pi, \mathscr{A} \in \Phi . P \models \diamond\mathscr{A} \stackrel{\triangle}{=} \exists P' \in \Pi .\ P' \models \mathscr{A} \wedge P \downarrow^* P'$

(x) *Sometime connective,* $\Diamond$, *is used for specifying temporal behavior of the processes on the basis provided by reduction relations* $(\rightarrow)$. *The relation* $\rightarrow^*$ *is reflexive transitive closure of reduction relation.* $\Diamond\mathscr{A}$ *is satisfied by processes which can evolve into a future process holding* $\mathscr{A}$. $\forall P \in \Pi, \mathscr{A} \in \Phi . P \models \Diamond\mathscr{A} \stackrel{\triangle}{=} \exists P' \in \Pi .\ P' \models \mathscr{A} \wedge P \rightarrow^* P'$

6.3. Ambient Calculus Model Checker for Security Policies

The general structure of the Ambient Calculus model checker is given in Figure 6.3. To benefit from existing methodologies we divide our problem into two sub problems as temporal model checking and spatial model checking. The temporal model checker is used for carrying out satisfaction process for the Sometime and Everytime connectives of Ambient Logic. The proposed model checking method generates all possible future states and build a state transition system based on the Ambient Cal-

culus process specification. After evaluation of Ambient Logic formula in each state, this state transition system is processed into a Kripke Structure (Definition 6.5) which is then given to temporal model checker. NuSMV [68] is used as a temporal model checker. Outline of the proposed algorithm for the model checking problem is presented below with respect to the components, inputs and outputs within the block diagram in Figure 6.3.

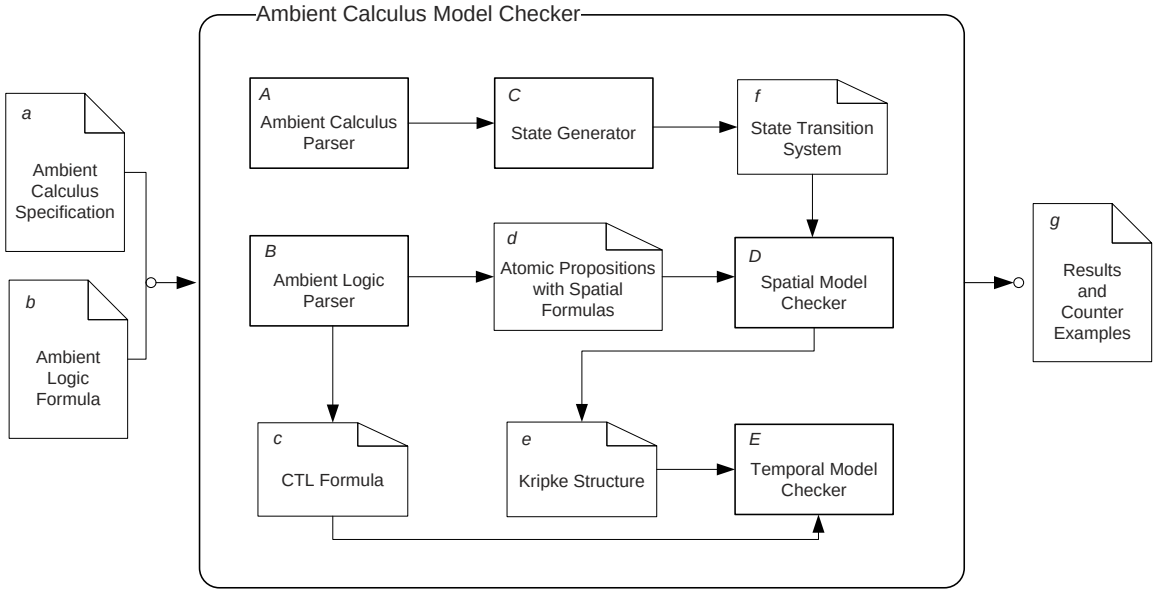

Figure 6.3. Block diagram of the Ambient Calculus Model Checker.

(i) The Ambient Calculus Parser (A) inputs Ambient Calculus Specification (a) and outputs the parsed specification to the State Generator (C).

(ii) The Ambient Logic Parser (B) inputs Ambient Logic Formula (b), defines atomic propositions with respect to spatial properties of Ambient Logic formula and generates the Atomic Propositions with Spatial Formulas (d) which includes the (atomic proposition-spatial modality) couples.

(iii) The Ambient Logic Parser (B) reduces Ambient Logic formula to CTL temporal logic formula (c) by replacing spatial modalities with atomic propositions.

(iv) The State Generator (C) generates State Transition System (f) of the Ambient Calculus Specification (a) with respect to reduction relations. This involves generation of initial state from given Ambient Calculus specification, generation of new states by applying available capabilities with respect to Ambient Calculus reduction relations and addition of new states to state transition system with

transition relation.

(v) The Spatial Model Checker (D) generates Kripke Structure (e) from State Transition System (f) and Atomic Propositions with Spatial Formula (d). This step involves the assignment of the values of the atomic propositions for each state of state transition system (labeling) by applying model checking for spatial modalities on ambient topology of the related state and the addition of a new state with its label (values of atomic propositions) to the Kripke Structure.

(vi) The Temporal Model Checker (E) generates NuSMV code from Kripke Structure (e) and CTL Formula (c). Then NuSMV is executed and Results and Counter Examples (g) are generated.

6.3.1. Ambient Topology and Spatial Formula Graphs

In [34], state information is represented with sets. In [35], calculus and logic information is represented as strings and algorithms are based on string operations. In the method proposed, Ambient Calculus specifications and logic formulas are represented as graphs. State information associated with a process specified in Ambient Calculus consists of static and dynamic properties. Static properties of state, called *ambient topology*, are the ambients and their hierarchical organization. The dynamic properties of the state are the capabilities and their dependencies on each other. Static and dynamic properties of an Ambient Calculus specification are kept in separate data structures.

Definition 6.2. Ambient Topology, $G_{AT} = (N_{AT}, A_{AT})$, *is an acyclic digraph where elements of set of nodes* $v \in N_{AT}$ *denotes ambients within the Ambient Calculus specification (elements of* Λ*) and arcs* $a \in A_{AT}$, $a = \{xy \mid x, y \in N_{AT}\}$ *denotes parent-child relation among ambients. The indegree of nodes* $deg^-(v) = 1$ *for any node (vertex)* v *whereas the outdegree of nodes* $deg^+(v) \in \mathbb{N}$.

The following defines *capability tree* which is a novel data structure used in our algorithm.

Definition 6.3. Capability Tree, $G_{CT} = (N_{CT}, A_{CT})$, *is an acyclic digraph where*

set of nodes $v \in N_{CT}$ *denotes capabilities and arcs* $a \in A_{CT}$, $a = \{xy \mid x,\ y\ \in N_{CT}\}$ *denotes priority relation among capabilities. Nodes contain the information about which ambient the capability is attached and which ambient the capability effects.* $deg^-(v) = 1$ *for any node* v, *whereas* $deg^+(v) \in \mathbb{N}$.

Graphs representing formulas are acyclic digraphs where nodes denote connectives and locations whereas arcs denote the operator-operand relation. There are multiple types of nodes and arcs in formula graphs because of the different structure of the Ambient Logic connectives.

Definition 6.4. *An* Ambient Logic formula, $G_F = (N_F, A_F)$, *is an acyclic digraph where*

- The set of nodes: $N_F = (N_L \cup N_{Binary} \cup N_{Unary} \cup N_{PC})$. N_L is the set of nodes representing ambients. Elements of N_L are labeled with elements of Λ. N_{Unary} is the set of nodes representing unary connectives ($\neg$, $\diamond$, $\Diamond$) at formulas. N_{Binary} is the set of nodes representing binary connectives, ($\vee$) at formulas. N_{PC} is the set of nodes representing parallel compositions at formulas.
- The set of arcs: $A_F = (A_{PC} \cup A_{Binary} \cup A_{Unary})$, where elements of A_{PC} represents parallel compositions, A_{Binary} represents binary connectives and A_{Unary} represents unary connectives of Ambient Logic formulas.
- $a_{pc} \in A_{PC} = x, y | x \in N_{PC},\ y \in (N_L \cup N_{Binary} \cup N_{Unary})$, $a_u \in A_{Unary} = (x, y \mid x \in (N_L \cup N_{Unary}), y \in N_{PC})$, $a_b \in A_{Binary} = (x, y | x \in N_{Binary},\ y \in N_{PC})$
- for $v \in N_F$, $deg^-(v) = 1$, for $v \in N_{PC}$, $deg^+(v) \in \mathbb{N}$, for $v \in N_{Unary}$, and $v \in N_L$, $deg^+(v)$ =1, for $v \in N_{Binary}$, $deg^+(v)$ =2..
- Elements of N_{PC} can have a special attribute to represent the $\top$ construct of the logic. If $\top$ attribute of a N_{PC} node is set to true, this means the parallel composition of process that the N_{PC} node stands for, includes the constant $\top$.

In Figure 6.4, an example for an ambient topology and a capability tree is presented. These structures make up the state information.

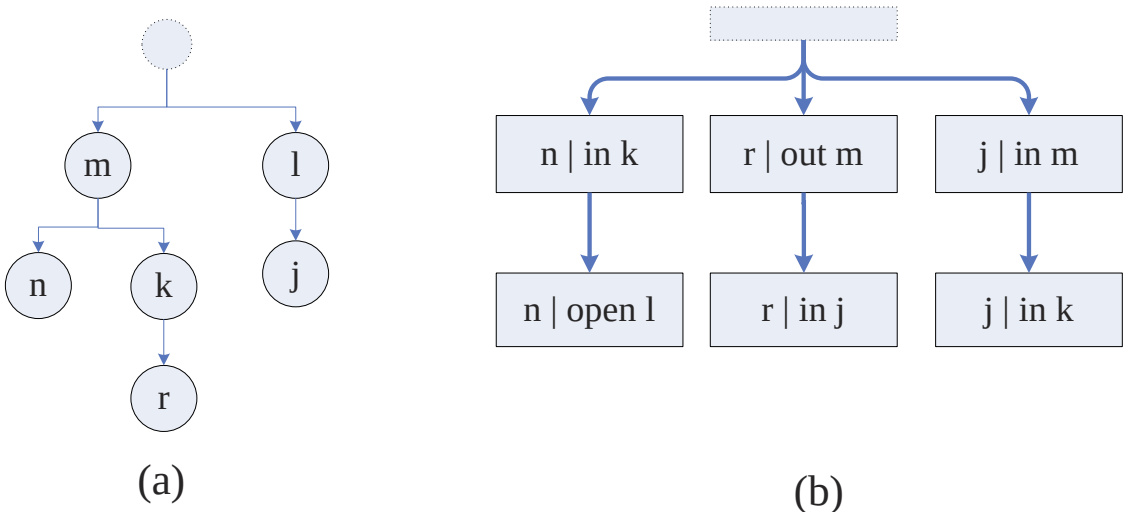

Figure 6.4. Internal representation of state information. Graph (a) is ambient topology of state and graph (b) is capability tree.

6.3.2. 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* ($\Diamond$) 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 :$n[\Diamond\mathscr{A}]$, $n[\Box\mathscr{A}]$, $\Diamond\mathscr{A}|\Diamond\mathscr{B}$, $\Box\mathscr{A}|\Box\mathscr{B}$, $\diamond\Box\mathscr{A}$,$\diamond\Diamond\mathscr{A}$. Such Ambient Logic formulas are not reducible to CTL formulas because they have temporal operators as sub-formulas of spatial operators. In the proposed algorithm Ambient Logic formulas must be restricted to temporal operators in higher levels compared to spatial operators. The restriction also brings an advantage. This restriction of Ambient Logic formulas provides the use of other CTL operators like AF or EG in a more straightforward way.

6.3.3. State Transition System Generation

In the proposed model checking algorithm the state transition system is generated from the initial model specification by executing capabilities in the Ambient Calculus specification. Since replication is excluded from specifications, the state transition system can be represented by an acyclic digraph where nodes represent states and edges represent the execution of a capability. For selection of the next capability to execute, some condition checks are carried out. These conditions are the location of the object ambient and the availability of the subject ambient. A capability can not be executed if the location of the object ambient for the capability is not the current location, if it is prefixed by another capability path, or the parent ambient of the subject ambient is prefixed by a capability path. In the proposed method, these conditions are checked each time a capability is to be executed.

In Figure 6.5, an example state transition system is presented for the Ambient Calculus process specification $P = m[in\ k.open\ l.n[]|k[out\ m.in\ j.r[]]]|l[in\ m.in\ k.j[]]$. In this figure, the edges represent Ambient Calculus capabilities which cause a transition.

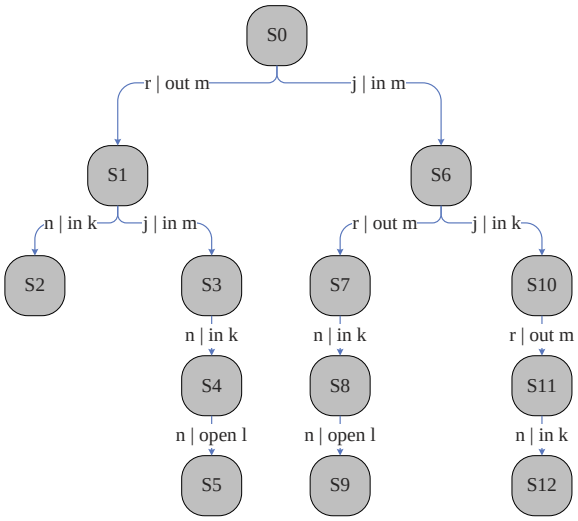

Figure 6.5. An example state transition system for an Ambient Calculus process specification.

We propose a new data structure to represent temporal behaviors. The use of this data structure, called *capability tree*, eliminates the need to check the availability of a subject ambient. Capability paths which are sequences of capabilities are organized as an acyclic digraph that represent the interdependencies of capabilities. Capability trees are built at parsing stage so no pre-processing is needed. The selection of the next capability to execute starts from the root of this graph. Our method guarantees that the capabilities of the parent processes are executed before the capabilities of child processes.

6.3.4. Checking Spatial Modalities

The basic element for building an Ambient Calculus model checker for Ambient Logic is to express and implement the satisfaction relation. In the proposed method, all the generated states must be checked with respect to the spatial formulas. Ambient Logic formulas are decomposed into a CTL formula and a set of spatial formulas by formula reduction. The ambient topology and the spatial formula graphs are inputs to the spatial model checker. The spatial model checking takes place before the generation of Kripke Structures.

Matching of an ambient topology and a spatial formula is a recursive procedure in which ambient topology nodes are assigned to formula nodes. Matching process starts by assigning the root of the ambient topology to the root of the spatial formula graph. Spatial formula nodes can forward the assigned ambient topology node to its children partially or completely in a recursive manner. Match process is successful when all nodes at ambient topology is matched to a spatial formula node. Match processes at different type of spatial formula nodes are different. Different match processes are introduced after auxiliary heuristic functions, which are explained below.

6.3.4.1. Heuristic Functions. Heuristic functions are used at matching the *Parallel Composition* ($|$) and *Somewhere* ($\diamond$) connectives. Former studies try to match every alternative while searching a match for these connectives. In our method, the number of these trials are reduced by the help of auxiliary heuristic functions. Some connectives

of Ambient Logic called wildcard connectives match different kinds of ambient topology. These connectives are used for matching ambients of ambient topology which are not expressed in formulas. The constant $\top$ of the logic matches any ambient topology assigned to it. *Negation* connective of the logic can be seen as another kind of wildcard connective. *Negation* matches any ambient topology unless the sub formula of the negation matches this ambient topology. Another wildcard property is the *Somewhere* connective. The parallel process of the parent ambient are neglected when searching sublocations. If the sublocation search is obtained by consecutive application of $\downarrow$ one or more times , the associated *Somewhere* connective gains a wildcard property. Function *wildcard* is a recursive function used for determining if a node of the formula graph has a wildcard property. The pseudocode of the *wildcard* function is given in Figure 6.6.

The ambients expected at sub-formulas of *Disjunction* and *Somewhere* connectives is not directly derivable. The *guessExpectedAmbients* function is a recursive function which returns a set of expected ambient combinations for a formula graph node. The returned set includes all possible ambient combinations expected by children of that node. The returned value is a set instead of a single ambient combination. The pseudocode of the *guessExpectedAmbients* function is given in Figure 6.7.

Function *findSublocation* is a recursive function used to find parent of an ambient at an ambient topology. The pseducode of the *findSublocation* function is shown in Figure 6.8.

6.3.4.2. Matching of Spatial Formula. In a match between an ambient topology and spatial formula graph, all nodes of ambient topology must be matched with a node of spatial formula graph. Some nodes of spatial formula graphs can forward the ambient topology nodes assigned to them to their children, while others match assigned ambient topology nodes directly. The proposed spatial model checking algorithm tries alternative assignments of a given ambient topology nodes over a given spatial formula graph. The proposed spatial model checking algorithm is recursive where matching process starts from the roots of a graph and continues to underlying levels. If a suitable

```
Procedure wildcard
Input: NODE is a formula graph node
Output: boolean constant
  if NODE is a location then
    return false
  end if
  if NODE is a disjunction then
    return wildcard(node.first_child) ∨ wildcard(node.first_child)
  end if
  if NODE is a somewhere then
    return true
  end if
  if NODE is a negation then
    return true
  end if
  if NODE is a parallel composition then
    if NODE has a ⊤ property then
      return true
    else
      for all child of NODE do
        if wildcard(child) = true then
          return true
        end if
      end for
    end if
  end if
  return false
  end procedure
```

Figure 6.6. Pseudocode of wildcard heuristic function.

```
Procedure guessExpectedAmbients
Input: NODE is spatial formula node
Output: set of string
  if NODE is a location then
    return NODE.name
  end if
  if NODE is a disjunction then
    return guessExpectedAmbients(node.first_child)
  end if
  if NODE is a somewhere then
    return guessExpectedAmbients(node.child)
  end if
  if NODE is a negation then
    return ∅
  end if
  if NODE is a parallel_composition then
    for all child of NODE do
      cart = cart × guessExpectedAmbients(child)
    end for
    return cart {the cartesian product of the elements of the returned values of
    guessExpectedAmbients for each child}
  end if
  end procedure
```

Figure 6.7. Pseudocode of guessExpectedAmbients heuristic function.

```
Procedure findSublocation
Input: WANTED is string, ROOT is ambient topology node
Output: RESPONSE is ambient topology node
  for all child of ROOT do
    if child.name == WANTED then
      return ROOT
    end if
  end for
  for all child of ROOT do
    RESPONSE = findSublocation(WANTED,child)
    if RESPONSE not == null then
      return RESPONSE
    end if
  end for
  return null
  end procedure
```

Figure 6.8. Pseudocode of findSublocation heuristic function.

match is found at the upper level then matching process continues to find matches in lower levels. The match process is regulated by the semantics of spatial formula graph nodes.

Matching process at nodes representing locations is according to Definition 6.1.5. The location nodes can be assigned only one ambient topology node. If the ambient topology node does not have the same name with the location node, match process for the location node fails. If the ambient topology node has the same name with the location node, location node assigns the children of ambient topology node to its parallel composition child. The result of the match is successful if the parallel composition child of location node succeeds to find a match between its children and the children of the assigned ambient topology node.

Matching process at the nodes representing *negation* connective is according to

Definition 6.1.2. 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 to find a match, match process for negation node fails. If no match between ambient topology nodes and children of the parallel composition is found, match process for negation node is successful.

Matching process at the nodes representing *disjunction* connective is according to Definition 6.1.3. These nodes can be assigned to a collection of ambient topology nodes. Nodes of type disjunction have two 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 to find to a match, match process for disjunction node is successful.

Matching process at the nodes representing *somewhere* connective is according to Definition 6.1.9. These nodes can be assigned to a collection of ambient topology nodes. A node of Somewhere connective can start matching its parallel composition node from any level of assigned ambient topology node collection. Match process of nodes for somewhere connectives try 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 which do not have any possibility to match.

Matching process at the nodes representing *parallel composition* connective is according to Definition 6.1.6. These nodes can be assigned to a collection of ambient nodes. Match process for parallel composition decomposes assigned ambient topology node collection into subsets which will be forwarded to the children of the parallel composition node. While there are exponentially many alternative decompositions, the number of these alternatives is reduced by the help of the *guessExpectedAmbients*

and *wildcard* functions.

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

Because there are ambient topology nodes which are not expected by any child node, these nodes must be assigned to one of the children of the node for parallel composition with wildcard property or neglected if the node for parallel composition has T (true) property. In the second phase, children of the parallel composition is evaluated for the wildcard property by the *wildcard* function. After determining the set of children with wildcard property, unexpected nodes of the ambient topology collection are assigned to 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. Match process succeeds if match processes of all children succeeds for a decomposition. In Figure 6.9, matching of an ambient topology and a spatial graph for a successful match are shown.

6.3.5. Generation of Kripke Structure

A Kripke Structure is a state transition system where states are labeled by the set of atomic propositions which hold in that state. Atomic propositions can be considered as the marking of system properties.

Definition 6.5. *Let AP be a non-empty set of atomic propositions. A Kripke Structure is a four-tuple; M = (S, S0, R, L) where S is a finite set of states, S0* $\subseteq$ *S is the set of initial states, R* $\subseteq$ *S*$\times$ *S is a transition relation, and L:* $S \rightarrow 2^{AP}$ *is a function that labels each state with the set of atomic propositions that are true in this state.*

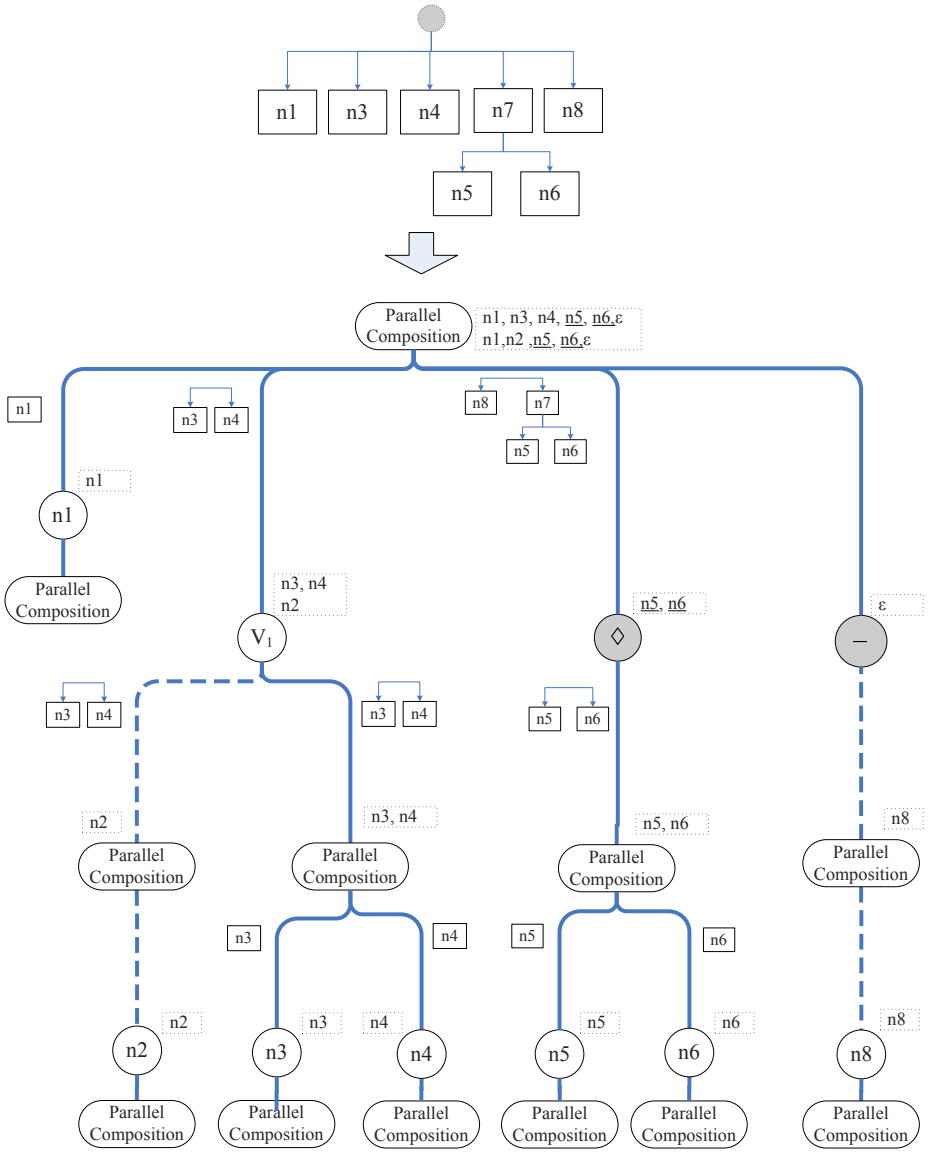

Figure 6.9. A match example for process P = $n1[] \mid n3[] \mid n4[] \mid n7[n5[] \mid n6[]] \mid n8[]$ and formula F = $n1[] \mid \{n2[] \ \vee \ \{n3[] \mid n4[]\}\} \mid \diamond\{n5[] \mid n6[]\} \mid \neg n8[]$. Graphs consisting of rectangle nodes are ambient topologies assigned to spatial formula graph nodes. Graphs consisting of circle nodes is spatial formula graph.[1]

The state transition data structure provides sets S, S0 and relation R of a Kripke Structure. The elements of the set of atomic propositions result from formula reduction. In formula reduction, spatial formulas are replaced with atomic propositions. The function L is generated by applying spatial model checking for each state in state transition data structure with respect to each spatial formula. Kripke Structure is obtained by attaching the values, resulting from spatial model checking, into the state transition system graph.

6.3.6. NuSMV Code Generation

The model checking algorithm explained above provides CTL formulas and a Kripke Structure. The next step is the generation of NuSMV code which is equivalent to the Kripke Structure and temporal logic formula. In the NuSMV specification, a variable *state* is used for specifying states in the Kripke Structure. Other variables used in NuSMV code generation is Boolean variables for representing atomic propositions. CTL formulas provided by the formula reduction step are then converted to NuSMV code according to CTL formula graph provided by formula reduction, where the Sometime ($\diamondsuit$) connective is represented as EF and Everytime ($\Box$) connective is represented as AG. The atomic propositions are converted into strings by their names.

6.3.7. Examples for Spatial Model Checking Algorithm

Let's consider the scenario and policy example presented in Section 3.3. When the Ambient Calculus specification is input to the model checker, a total of 53 states are generated. One Atomic Proposition (AP) is generated, where AP=$\diamond$ { $\diamond$ *Host2*[$\diamond$ {*Data1*[T]|*Data2*[T]}] | T}. A part of the execution of the algorithm is presented in Table 6.3. Only the initial and the last two states are shown. For each state an action is executed to produce a new spatial state. For state 53 the spatial model checking algorithm matches the spatial formula AP to the current state of World.

For the purpose of evaluation of the performance of the algorithm, we have checked three Ambient Calculus specifications with respect to two Ambient Logic for-

Table 6.3. Part of output generated by the spatial model checker for the example policy presented in 3.3.

State	Spatial state of World	AP	Action
0	Domain1[User1[]|Host1[File1[Data1[]]]]| Domain2[Host2[User2[]|File2[Data2[]]]]	F	User2[out Host2]
52	Domain1 [User1 [Host1 [File1[]]]| Domain2 [Host2 [File2 [User2[]|Data2 []|Data1 []]]	F	User2[out File2]
53	Domain1 [User1 [Host1 [File1[]]]| Domain2 [Host2 [File2 [Data2 []|Data1 []]|User2[]]	T	-

mulas. Ambient Calculus specifications and Ambient Logic specifications are presented in Appendix B. Each Ambient Calculus specification models a multi domain network where domains, host, user, and files are modeled as ambients. Ambient Logic formulas represent different properties of these models.

The given specifications represents a scenario involving mobility among multiple domains. The case study includes three domains *Domain A*, *Domain B*, *Domain C*, four hosts *Host 1*, *Host 2*, *Host 3*, *Host 4*, three users *User 1*, *User 2*, *User 4* and four 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 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*, read *File 3* and *File 4*. The hosts *Host 2* and *Host 3* are accessible and readable by users of *Domain C*. *Host 1* in *Domain A* is accessible by *DomainB* 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 security policy rules in this example, *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 lead to an unintended information flow: *User 2* can read *File 1*, 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, which will be discussed in the next section.

6.3.8. Complexity and Performance Analysis

6.3.8.1. Time Complexity. Time complexity of state transition system generation is dependent on the number of capabilities. The execution of a capability results in a single future state. In the worst case, all capabilities are independent. Independence of capabilities implies that capabilities are in a sequence or they operate on different ambients. 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. As a result, where n is the number of capabilities in the Ambient Calculus specification,

the time complexity of generating state transition system in worst case is $O(n!)$ and more specifically

$$\sum_{k=0}^{n} \frac{n!}{k!} \tag{6.1}$$

The time complexity of checking spatial modalities are dependent on the type and number of the connectives of the spatial formulas. 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 two 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 linear with the cost of the match process of | connectives, which is calculated as follows. Let a_{ne} be the number of topmost ambients of the ambient topology which are not expected by the heuristic functions, d_w the number of disjunctions which have wildcard property in the | connective, *not* the number of negations in the | connective, sw_w the number of $\diamond$ connectives which have wildcard property in the | connective, G the cost of calculating *guessExpectedAmbients* function and W the cost of calculating *wildcard* function.

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

$$2^{d_w+d} \tag{6.2}$$

Unexpected nodes of ambient topology can be assigned only to the formula nodes which has a wildcard property. The number of different assignments of unexpected

nodes are

$$a_{ne}^{(sw_w+not+d_w)} \tag{6.3}$$

The overall time cost of the match process is given in Formula 6.4, which is the product of Formula 6.2 and Formula 6.3 with addition of the costs of the auxiliary functions:

$$2^{d_w+d} \times a_{ne}^{(sw_w+not+d_w)} + G + W \tag{6.4}$$

The time complexity of the match process, corresponding to the calculated time cost in Formula 6.4, is given according to the O notation in Formula 6.5. The time complexity is exponential with the number of ambients.

$$O(a_{ne}^{(sw_w+not+d_w)}) \tag{6.5}$$

In contrast, when the brute force search is used for decomposing Ambient Calculus specifications, the time complexity is calculated as in Formula 6.6, where $a = a_{ne}+a_e$ is the total number of topmost ambients in the ambient topology, including (a_e), number of ambients expected by the heuristic functions, l is the number of locations in the | connective, sw is the number of $\diamond$ connectives which do not have wildcard property in a | connective and d is the number of disjunctions which do not have wildcard property in | connectives:

$$O((a)^{(sw_w+sw+l+not+d+d_w)}) \tag{6.6}$$

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. Time cost of finding a match for the $\diamond$ connective is the sum of cost of the function *findSublocation* and cost of the match process for | connective. Let PC be the cost of match process of | connective child for the $\diamond$ connective,F the cost of *findSublocation* function, a the number of the ambients of the ambient topology. The

overall time cost of the match process for $\diamond$ connective is given with Formula 6.7:

$$F + PC \tag{6.7}$$

Time cost of brute force search for finding a matching for $\diamond$ connective is given with Formula 6.8:

$$a \times PC \tag{6.8}$$

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

The match processes for | and $\diamond$ connectives are dominant for the time complexity of the spatial model checking process. The cost of the overall match process is reduced by the proposed algorithm. The proposed algorithm and brute force search for match process for | connective both have exponential complexity, given respectively in Formula 6.5 and Formula 6.6. Since $(a_{ne} + a_e)^{(sw_w+sw+l+not+d+d_w)} > a_{ne}^{(sw_w+not+d_w)}$, the cost of the match process is significantly reduced by use of heuristics. The complexity of the match process for the $\diamond$ connective for both the proposed algorithm and brute force search are linear. However the actual cost is reduced from Formula 6.8 to Formula 6.7.

6.3.8.2. Space Complexity. Proposed algorithm builds a state transition system in a depth-first manner. The maximum depth of the state transition system is equal to the number of capabilities. Therefore, the space complexity of the space generation is $O(n)$, where n is the number of capabilities. When checking spatial modalities, the space needed is equal to the size of the formula, which is dependent on the number of connectives of the formula. Therefore, the space complexity of checking spatial modalities is $O(c)$, where c is the number of the connectives in the formula.

Table 6.4. Properties of Ambient Calculus specifications.

Specification	**Num. Ambients**	**Num. Capabilities**	**Num. States**
Spec 1	16	32	560
Spec 2	16	39	33123
Spec 3	16	37	628527

6.3.8.3. Performance Analysis based on Example Specifications. The proposed model checking algorithm has been implemented in the Java language. In this section, the implementation is tested with example formal specifications which is presented in Appendix B and explained in Section 6.3.7. The example specifications include three Ambient Calculus specifications and two formulas. Each Ambient Calculus specification models a multi domain network where domains, host, user and files are modeled as ambients. Formulas represent different properties of these models.

Tests are carried out on one of the nodes of a cluster. The cluster itself has 8 nodes with 2.93 GHz CPU 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 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.

The properties of Ambient Calculus specifications are shown in Table 6.4. All the specifications consist of the same set of ambients where their starting ambient topology is the same. The main difference between them is the type and the number of the capabilities. Because of this property, the future evolution of the models vary dramatically.

Three Ambient Calculus specifications are evaluated with respect to two Ambient Logic formulas. The time and memory cost for state transition system is shown in Table 6.5, for 8 MB and 1GB heapsizes of the Java Virtual Machine. Since memory management of the Java Virtual Machine (JVM) includes a garbage collector, the heapsize effects performance of Java applications. For this reason the model checker

Table 6.5. State transition system generation cost.

Specifications	**Heap Size**	**Time (sec)**	**Memory (KB)**
Spec 1	1 GB	1.039	246478
	8 MB	1.234	6228
Spec 2	1 GB	12.453	350504
	8 MB	18.120	7734
Spec 3	1 GB	154.897	362384
	8 MB	241.592	7911

process has been executed with various heapsizes. The default heapsize of Java VM on the cluster was 8MB. The heapsize has been increased to evaluate the effect of increase in heapsize to performance. The best performance has been achieved at 1GB. For the case study, the heap sizes over 1GB did not result in an increase in performance.

Table 6.6. Performance results for spatial model checking.

	Formulas			
	Formula 1		**Formula 2**	
Specifications	**Time (sec)**	**Memory (KB)**	**Time (sec)**	**Memory (KB)**
Spec 1	1.265	262208	1.520	262208
Spec 2	15.474	350872	17.134	351080
Spec 3	172.289	362304	199.815	375352

In Table 6.6, the performance results are shown for spatial model checking with a 1 GB heapsize of the Java Virtual Machine. The time cost of NuSMV temporal model checking is shown in Table 6.9. The cost of state transition system generation outweighs spatial model checking for both time and space cost. A significant result of these case studies is that 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 628527 states with memory consumption under 8 MB. Time cost of NuSMV model checking significantly increases as the number of states increases. Size of the generated NuSMV code grows linearly as the number of states increases. However NuSMV can

not process the generated code for the specification with 628527 states; it terminates with segmentation fault because of the size of the generated NuSMV code.

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 two given formulas in the case study. The *branching factor* property is related with 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 depth of the syntax tree which represents a formula. The properties of given Ambient Logic formulas in the case study are presented in Table 6.7.

Table 6.7. Properties of formulas.

Formula	**Branching factor**	**Depth**
Formula 1	1	4
Formula 2	2.6	3

We compare the use of heuristics against brute force search for exploration of spatial modalities. Since former studies which use brute force search [69, 35] 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 6.8. As seen from these results, the proposed algorithm produces better performance results when the branching factor of the formula increases.

Table 6.8. Performance results for spatial model checking with brute force search.

	Formulas			
	Formula 1		**Formula 2**	
Specifications	**Time (sec)**	**Memory (KB)**	**Time (sec)**	**Memory (KB)**
Spec 1	1.634	262208	1.829	262208
Spec 2	15.334	350672	18.405	353392
Spec 3	171.812	364696	201.765	375160

Time cost of NuSMV model checking significantly increases as the number of states increases. Size of the generated NuSMV code grows linearly as the number of states increases. However NuSMV can not process the generated code for the specification with 628527 states; it terminates with segmentation fault because of the size of the generated NuSMV code.

Table 6.9. Performance results of NuSMV with generated code.

Specifications	**Formulas**	
	Time for Formula 1 (sec)	**Time for Formula 2 (sec)**
Spec 1	0.126	0.135
Spec 2	463.918	615.361
Spec 3	N/A	N/A

7. THEOREM PROVING FOR SECURITY POLICIES IN FPFM

Theorem provers are automated procedures that can be used to check whether a given formula F, which is named the *goal*, is a logical consequence of a set of formulas N, which is called the *theory*. Proof checking consists of the automated verification of theories by full formalization of the primitive notions of definitions, axioms and the proofs. The definitions are checked for their well-formedness and the proofs are checked for their correctness according to a given logic. For theorem proving, one needs to choose a logic, which is the basis of all formalization. Logics such as classical logic, intuitionistic logic, first, second, higher order logic may be used for formalization.

In this thesis, we use the *Calculus of Inductive Constructions* (*CIC*) for the basis of theorem proving approach for security policies. *CIC* is a kind of higher-order, intuitionistic logic which is implemented by the *Coq Proof Assistant*. CIC is based on type theory, which is another name for higher-order logic. Type theory presents a powerful formalism which covers both *computation* and *proof*. Furthermore, CIC makes use of the "*proofs as programs*" paradigm which is known as the *Curry-Howard Isomorphism*. In this paradigm, a proposition is a type and a proof is a computation of a function. Extraction of programs from proofs is possible thanks to the Curry-Howard Isomorphism.

In type theory, the formalization of the proof checking for a statement A is $\Gamma \vdash_T p : A$, which is equivalent to checking whether $Type_\Gamma(p) = A$. When using an interactive proof assistant based on type theory such as Coq, the proof terms are generated by the proof development system by interacting with the user. The proof term is type checked by the proof checker and compared with the original goal. If the type checking is successful then p is a proof of A. Otherwise, proof term p does not have the type of A and the type checker returns false. The equality $=$ in this context is decidable. Therefore the concepts of interactive theorem proving based on type theory may be summarized in Table 7.1.

Table 7.1. The concepts of interactive theorem proving based on type theory.

Concept in theorem proving	**Concept in type theory**
provability of formula A	inhabitation of type A
proof checking	type checking
interactive theorem proving	construction of a term of a given type

Proof assistants, synonymously interactive theorem provers, are slightly different than automated theorem provers, since the proof of the theorems are generated by the guidance of the user. A proof assistant is a combination of proof development system and a proof checker. The proof development system aids the user by providing language constructs for specification and algorithms for verification of theorems. The user controls the theorem development process and may define data structures and executable functions. Theorems about the data structures and the functions may be proved with the help of the proof assistant.

We introduce the Coq Proof Assistant in Section 7.1. In the remaining sections, we present formal specification of security policies in FPFM with the Calculus of Inductive Constructions. Through Sections 7.2 to 7.5, we present formal specifications related to data sets, authorization terms and hierarchies related to security policies in FPFM. In Section 7.6 we present conflict checking of security policies with the Coq theorem prover, based on a subset of the security policy model in FPFM.

7.1. The Coq Proof Assistant

Coq is a proof assistant for expressing specifications and developing programs that fulfil these specifications. It belongs to a family of interactive theorem provers such as Mizar, Isabelle, Lego and HOL. It is based on a variation of typed Λ-calculus, the Calculus of Inductive Constructions (CIC in short). CIC includes the ordinary logical operators $\wedge, \vee, \neg, \rightarrow$. Quantifications are typed in the form $\forall x : T\ P\ x$, where x is a quantifier (variable), T is a type and P is a proposition. Due to the Curry-Howard isomorphism, $P \rightarrow Q$ means both P implies Q and a total function which computes a proof of Q from a proof of P. Propositions have the type `Prop`. $P \rightarrow Q$

also defines a type expression. An expression $A_1 \to A_2 \to \ldots A_n \to B$ denotes the type of a function which has n arguments of types $A_1 \ldots A_n$ and which returns a result of type B. Data types such as the type for integers, `nat`, are members of the `Set` type. This allows building polymorphic functions whose type depends on the input arguments. For example the identity function is defined as `Id` $x = x$, whose type is $\forall X :$ `Set` $X \to X$.

Inductive types provide means to define data types from constructors. The general form of an inductive type is $\{x : S | P\ x\}$, which defines a set of elements x from S which satisfy the predicate $P\ x$. An *inductive definition* consists of an exhaustive enumeration of constructors of the type to be defined together with their respective signatures. For example, the types `bool` and nat are defined in Figure 7.1.

```
Inductive bool:Set:=true:bool | false:bool

Inductive nat:Set:=0:nat | S:nat→nat
```

Figure 7.1. Inductive definitions for the types bool and nat.

A predicate may also be defined in an inductive form, similar to Prolog. To define an predicate, one enumerates clauses that define sufficient conditions for the predicate to be satisfied. For example, to define a predicate "to be a sorted list", one needs to consider three clauses. This predicate is defined in Coq as in Figure 7.2.

(i) the empty list is sorted

(ii) every list with only one element is sorted

(iii) if a list *n::l* obtained by concatenation of n to l is sorted, a list *p::n::l* is sorted if $p \leq n$.

```
Inductive sorted:list Z→Prop:=
sorted0:sorted(nil)
sorted1:∀z : Z, sorted (z :: nil)
sorted2:∀z1, z2 : Z, ∀l : list Z, z1 ≤ z2 ⇒ sorted(z2 :: l) ⇒ sorted(z1 :: z2 :: l)
```

Figure 7.2. The definition of a predicate in Coq.

Pattern matching is used in Coq to describe functions that perform a case analysis

on the value of an expression whose type is an inductive type. Where t is an expression with constructors $c_1, c_2, \ldots$, the general form of a pattern matching construct is of the form `match` t `with` $c_1 \Rightarrow e_1 \mid c_2 \Rightarrow e_2 \ldots \mid c_t \Rightarrow e_t$ `end`. The value of this expression is e_1 if the value of t is c_1, and so on, finally e_t if the value of t is c_t.

For the proof of a proposition, the user constructs a function which computes a proof. A proposition may be a hypothesis, axiom, lemma or theorem, which are declared by keywords `Hypothesis`, `Lemma`, `Axiom` or `Theorem`. All propositions have the type `Prop`. A *goal* is the pairing of the local context Γ and a type t which is well formed in this context. A goal is of the form "$E, \Gamma \stackrel{?}{\vdash} P$", which expresses that a proof of P that should be well-formed term t in the environment E and context Γ should be provided. The proof of propositions are handled in Coq using *tactics*. Tactics are commands that can be applied to a goal. If g is a goal that needs to be proved and $g_1, \ldots g_k$ are the available goals, a tactic is associated with a function that constructs a solution of g from the solutions of $g_1, \ldots g_k$.

7.2. Data System Definitions for Security Policies

In the following sections we present a formalization of FPFM security policy model in Calculus of Inductive Constructions. This security policy models excludes location formulas, which are modelled within the spatio-temporal model checking approach. The object types in FPFM are defined using an inductive type. The set of object types is fixed and includes file, directory, application, database, portable, server, client and message types. File, directory, application and database types represent application objects. In contrast, portable, server, client and message types represent network objects.

The object names are finite set defined by the user specific to a data system. They are the names of resources in a data system. Object Type Hierarchy (OTH) relation is then defined by mapping each object name to an object type. This is essentially a relation implemented as a list that defines the elements of the relation in the style R={(a,b), (c,d), (a,e)}.

The next step in specifying a Data System is Role Hierarchy. In order to define the RH, one first needs to define the users. The user names are finite sets whose elements are user-definable and in CIC they are specified as inductive types. In the Coq specification of Role Hierarchies, The Roles are defined similar to Users, as an Inductive type with pre-defined elements. The members of the Role hierarchy is defined through a relation whose elements belong to the Cartesian product of the set Role with itself. The set of actions is defined by the inductive type Actions.

Membership for the relations defined for this data system are specified as predicates as follows. *Descendant_Role* predicate is true if a given role is descendant (or specialization) of another given role. And *Object_is_Type* predicate is true if an object identified by its object name is of a given object type. The data set definitions are given in Figure 7.3.

```
Inductive Object_Type : Set :=
 | File | Directory | Application | Database | Port | Server | Client | Message.
Inductive Object_Name : Set
Definition Object_Name_Object_Type (o: Object_Name) : Object_Type
Inductive User_Name : Set Inductive Role : Set Inductive Actions : Set:=
 | read | write | execute | send | receive | enrol | login | logout.
```

Figure 7.3. Data system definitions for security policies.

7.3. Formal Specification of Data System for Security Policies

All of the elements of the data system are a member of the inductive type Entity. This type is like a placeholder for all data system element types. An object is specified as "object *object_name*", e.g. "object research_project_db". Similarly, a user is specified as "user *user_name*", an object type as "object_type *type"*, and a role as "role *role_name*". The set of entities is specified using the *Ensemble* type in Coq. *Entity_Set* is a set that includes all members of the type Entity i.e. all the elements of the data system. The definition of Entity is given in Figure 7.4.

Next we define some predicates to check the element type of a specific data system

```
Inductive Entity : Set :=
  object : Object_Name→ Entity
| object_type: Object_Type → Entity
| user : User_Name → Entity
| role : Role→ Entity.

Definition ENTITY_SET := Ensemble Entity.
Definition Entity_Set : ENTITY_SET := Full_set Entity.
```

Figure 7.4. The inductive type Entity.

entity. These predicates would tell whether a given entity is an object, user, object type, user group or role. In Figure 7.5, we give the specification of the predicate Is_Object that becomes true if the given entity is an object. Other predicates are defined in a similar fashion.

In order to be able to map objects, users, user groups and roles to entities, a mapping between the types that define these system elements is needed. This mapping is provided through Coercion feature of Coq. Coercion implements the inheritance mechanism for types in Coq. The following lines of Coq specification in Figure 7.5 enables an object name to be handled as an entity. An object name *on* becomes an entity with the mapping *object on*. The other types belonging to user, object type, user group and role are similarly mapped to entities by defining coercions.

```
Definition Is_Object (e: Entity) : Prop :=
match e with
  object _ ⇒ True
  | _ ⇒ False
end.
Definition O_E (on: Object_Name) : Entity := object on.
Coercion O_E : Object_Name >-> Entity.
```

Figure 7.5. Specification of Is_Object predicate.

7.4. Formal Specification of Authorization Policy

According to the authorization framework, authorization subjects may be users, user roles or roles. In order to define a user, user group or a role as an authorization subject, an inductive type is used with the constructors that take a user name, user group or a role and return an authorization subject. The set union of the set of roles, set of user names and set of user groups make up the set of authorization subjects (Authorisation_Subject_Set). The specification of authorization subjects is given in Figure 7.6.

```
Inductive AuthorisationSubject: Set :=
 | user_as : User_Name → AuthorisationSubject
 | user_group_as : User_Group → AuthorisationSubject
 | role_as : Role → AuthorisationSubject.
Definition Authorisation_Subjects := ENTITY_SET.
Definition Authorisation_Subject_Set := Union Entity Role_Set
(Union Entity User_Name_Set)
```

Figure 7.6. Specification of authorization subjects.

The next step is mapping user, user group and roles types to authorization subject types. This is achieved through coercions that take a user, user group or role as input and convert it to the type AuthorisationSubject by use of its constructors. Every authorization subject is also a system entity, therefore there is a type coercion that maps an authorization subject to a system entity is also necessary. The function AS_E which is defined as coercion takes an authorization subject and returns the corresponding entity. Coercions for authorization objects are similarly defined. The specification is given in Figure 7.7.

The authorizations are specified as given in Figure 7.8. First authorization signs are specified as a set that contains the signs plus and minus. Second, the authorization signs together with actions define a record type for signed actions. An authorization term is a record that contains an authorization object, an authorization subject and a signed action. An authorization term is defined as “authorisation ao as sa” where

```
Definition U_AS (un: User_Name): AuthorisationSubject := user_as un.
Coercion U_AS : User_Name >-> AuthorisationSubject.
Definition R_AS (r: Role) : AuthorisationSubject := role_as r.
Coercion R_AS : Role >-> AuthorisationSubject.
Definition AS_E (aus: AuthorisationSubject) : Entity :=
match aus with
  user_as u ⇒ user u
 |role_as r ⇒ role r
end.
Coercion AS_E : AuthorisationSubject >-> Entity.
```

Figure 7.7. The mapping of user, user group and roles types to authorization subject types.

ao:Authorisation Object, as: AuthorisationSubject and sa: SignedAction. An example authorization term is "authorisation research_project_1_db johnd (Signed_Action plus read)".

The function *Negative_Sign* returns the negative of a given sign. The function utilizes the pattern matching construct *match* in Coq to determine the value of the input sign. It returns *minus* if *plus* is input and returns *plus* when *minus* is input. The function *Negative_Signed_Action* returns the negatively signed action for a given signed action (e.g. returns *Signed_Action plus read* when *Signed_Action minus read* is given). It simply returns the equivalent action with a negative sign using the *Negative_Sign* function. The pattern-matching construct with a single case deconstructs a record type for reading its fields. The specification of these functions are given in Figure 7.8.

The *Negative_Signed_Authorization* function returns the negatively signed authorization term for a given term, i.e. (*si, oj,* (-/+)*at*) is returned for a given term (*si, oj,* (+/-)*at*).

An authorization policy is specified as a set whose elements are authorization terms. This specification uses the set type in Coq, which implements sets as lists. The

```
Inductive Authorisation_Signs : Set :=
| plus | minus.

Record SignedAction : Set := Signed_Action
{Sign: Authorisation_Signs;
 Action: Actions}.

Definition Negative_Sign (sign: Authorisation_Signs) : Authorisation_Signs :=
match sign with
| plus ⇒ minus
| minus ⇒ plus
end.

Definition Negative_Signed_Action (sa: SignedAction): SignedAction :=
match sa with Signed_Action sign action ⇒ Signed_Action (Negative_Sign sign)
action end.

Definition Signed_Actions := set SignedAction.
Parameter SA : Signed_Actions.

Record Authorisation_Term :Set := authorisation
{o: AuthorisationObject;
 s: AuthorisationSubject;
 sa: SignedAction}.

Definition Authorisation_Policy := set Authorisation_Term.
Definition AUTH : Authorisation_Policy
```

Figure 7.8. Specification of authorization terms and authorization policy.

append operator (::) concatenates an element or a list to another list. The definition of authorization policy ends with the special list constructor nil. The definition for an authorization policy is provided in Figure 7.9.

7.5. Formal Specification of Hierarchies in Security Policies

This section presents the generic formal definition of hierarchies. Data system specific membership relations had been defined previously. Here, we present the generic

```
Definition AUTH : Authorisation_Policy :=
 (authorisation project_1_db alib (Signed_Action plus read)) ::
 (authorisation project_1_db alib (Signed_Action minus write)) ::
 (authorisation project_1_db grad_student (Signed_Action plus write)) ::
 (authorisation mail_server uni_user (Signed_Action plus read)) ::
 (authorisation mail_server uni_user (Signed_Action plus write)) ::
 (authorisation mail_server non_uni_user (Signed_Action plus read)) ::
 (authorisation email Member (Signed_Action plus send)) ::
 (authorisation email Member (Signed_Action plus receive)) ::
 (authorisation report Project_member (Signed_Action plus write)) ::
 (authorisation Server Admin_staff (Signed_Action plus administrate)) ::
 nil.
```

Figure 7.9. An example authorization policy specification.

formal specification for hierarchies.

The Object-Type Hierarchy (OTH) depends on the relation $\leq_T$ (specified by *Lt_eq_object_rel*) which specifies that a given object is a member of an object type. This is determined by membership to the relation *Object_is_Type* in the data model. The predicate *Lt_eq_object_type* is true if two supplied entites are a member of the *Object_is_Type* relation. Two entities are a member of the $\leq_T$ relation if the first entity is an object, the second entity is an object type and they satisfy the *Lt_eq_object_type* predicate. The specification of the $\leq_T$ relation is given in Figure 7.10.

The Role Hierarchy (RH) is based on the relation $\leq_R$. The predicate Lt_eq_role is true if two roles are member of domain specific relation Descendant_Role. Since this hierarchy consists only of roles, the inverse of this relationship is specified by the predicate Lt_eq_role_inv. The relation $\leq_R$ is specified by Lt_eq_role_rel which states that two roles are a member of $\leq_R$ if they satisfy the Lt_eq_role predicate. The specification of the $\leq_R$ relation is given in Figure 7.11.

After the specification of $\leq$ relations the next step is specifying hierarchies for-

```
Definition Lt_eq_object_type (o: Object_Name) (ot : Object_Type) :Prop :=
In Object_Prod Object_Type_Map (o, ot).

Definition Lt_eq_object_rel (o_a: Entity) (o_b: Entity) : Prop :=
match o_a with
 object o ⇒
   match o_b with
     object_type ot ⇒ Lt_eq_object_type o ot
     | _ ⇒ False
   end
  | _ ⇒ False
end.
```

Figure 7.10. The specification of the $\leq_T$ relation.

```
Definition Lt_eq_role_rel (r_a : Entity) (r_b : Entity) : Prop :=
match r_a with
  role ra ⇒
    match r_b with
      role rb ⇒ Lt_eq_role ra rb
      | _ ⇒ False
    end
   | _ ⇒ False
end.
```

Figure 7.11. The specification of the $\leq_R$ relation.

mally. There are 5 types of hierarchies which are defined with the inductive type Hierarchy_Types. An hierarchy is defined by an hierarchy type, a primitive set of entities upon which hierarchy is constructed, a secondary set of entities which defines the relations with the primitive set, and the $\leq$ relation between entities the type of which changes according to the primitive and secondary sets.

From this definition, OTH is a hierarchy whose type is O_T_H, whose primary set is the set of objects, whose secondary set is the set of object types, and which is based on $\leq_T$ relation. RH hierarchy which is of type R_H, has the role set as both primary and secondary set and is based on the $\leq_R$ relation. The specification of OTH and RH are given in Figure 7.12.

```
Inductive Hierarchy_Types : Set := | O_T_H | R_H.

Record Hierarchy : Type := hierarchy
                {HierarchyType: Hierarchy_Types;
                PrimitiveSet : Set;
                AggregateSet: Set;
                Lt_eq : Relation Entity}.

Definition OTH : Hierarchy :=
hierarchy O_T_H Object_Name Object_Type Lt_eq_object_rel.

Definition RH : Hierarchy := hierarchy R_H Role Role Lt_eq_role_rel.
```

Figure 7.12. The specification of OTH and RH.

7.6. Conflict Checking of Authorization Policies with the Coq Theorem Prover

Conflict checking of security policy rules determines *modal conflicts* in security policies. A modal conflict is a conflict where a subject is given both a permission and a denial for a given object. In this section, for the purpose of conflict checking of security policy rules, we use a simple model for authorization policy rules. Instead of using the identifiers of subjects and objects, we use integers for identification. Additionally, we assume that there is no role hierarchy. We also omit conditions and location formulas

from this analysis.

7.6.1. Authorization Policy Model

The specification of authorization policy model for conflict checking is given in Figure 7.13. Authorization subjects and objects are specified with the types *AuthorizationSubject* and *AuthorizationObject* as "subject i" and "object j" where i,j are natural numbers. We omit the names of subjects and objects since they are irrelevant for specification and verification purposes. The set of actions is extendible, yet a minimal set used here includes the read, write and execute actions. This is a finite set with known elements specified using the inductive type *Actions*.

Authorization signs specified with the type *Authorization_Signs* define permissions for an action. *Plus* (+) denotes explicit permission and *minus* (-) denotes explicit denial. A signed action is an action associated with a sign and is of the form *Signed_Action sign action* (such as *Signed_Action plus read*). This utilizes the *Record* type in Coq, which is similar to the "struct" construct in C language, with the difference that all instances must be well-typed and values must be determined at the time of declaration. The *Sign* field corresponds to an authorization sign, whereas *Action* field is a member of the inductive type *Actions*.

The function *Negative_Sign* returns the negative of a given sign. The function utilizes the pattern matching construct match in Coq to determine the value of the input sign. It returns minus if plus is input and returns plus when minus is input. The function *Negative_Signed_Action* returns the negatively signed action for a given signed action (e.g. returns *Signed_Action plus read* when *Signed_Action minus read* is given). It simply returns the equivalent action with a negative sign using the *Negative_Sign* function. The pattern- matching construct with a single case deconstructs a record type for reading its fields.

An authorization term is the basis of an authorization rule. This is a triplet of the form (s_i, o_j, sa) where s_i : $AuthorizationSubject, o_j$: $AuthorizationObject$ and sa :

```
Inductive Entity : Set := | subject_e : nat → Entity | object_e : nat → Entity.
Inductive AuthorisationSubject : Set := | subject : nat → AuthorisationSubject.
Definition AS_E (aus: AuthorisationSubject) : Entity :=
match aus with subject u ⇒ subject_e u end.
Coercion AS_E : AuthorisationSubject >-> Entity.
Inductive AuthorisationObject : Set := | object : nat → AuthorisationObject.
Definition AO_E (auo: AuthorisationObject) : Entity :=
  match auo with object u ⇒ object_e u end.
Coercion AO_E : AuthorisationObject >-> Entity.
Inductive Actions : Set := | read | write | execute.
Inductive Authorisation_Signs : Set := | plus | minus.
Record SignedAction : Set := Signed_Action
 {Sign: Authorisation_Signs; Action: Actions}.
Record Authorisation_Term : Set := authorisation s : AuthorisationSubject; o :
AuthorisationObject; sa : SignedAction}.
Definition signedaction (a: Authorisation_Term) : SignedAction :=
 match a with authorisation s o sa ⇒ sa end.
Definition Negative_Sign (sign: Authorisation_Signs) : Authorisation_Signs :=
  match sign with | plus ⇒ minus | minus ⇒ plus end.
Definition Negative_Signed_Action (sa: SignedAction): SignedAction :=
  match sa with
    Signed_Action sign action ⇒ Signed_Action (Negative_Sign sign) action end.
Definition Negative_Signed_Authorisation (a_u: Authorisation_Term) :
Authorisation_Term :=
  match a_u with
  authorisation a_s a_o s_a ⇒ (authorisation a_s a_o (Negative_Signed_Action
s_a))
  end.
Definition Signed_Actions := set SignedAction.
Parameter SA : Signed_Actions.
```

Figure 7.13. The specification of authorization policy model for conflict checking.

SignedAction. An authorization term determines whether policy explicitly allows or denies an action on an object by a subject. An authorization term is specified as *authorization s o sa*. For example, an authorization term $(s_1, o_2, +read)$ is specified as *authorization (subject 1) (object 2) (Signed_Action plus read)*. The authorization term is defined with the record *Authorization_Term*. The *Negative_Signed_Authorization* function returns the negatively signed authorization term for a given term. $(s_i, o_j, (-/+)a)$ is returned for a given term $(s_i, o_j, (+/-)a)$.

An authorization policy is a set of authorization terms. This definition uses the set type in Coq, which implements sets using lists. The *::* infix operator on lists is appends an element to the head of the list. A list is then specified as *a::b::c::nil* where *nil* marks the end of a list. The set *AUTH* defines a security policy. The Coq code for the example authorization policy for conflict checking is given in Figure 7.14. This definition corresponds to the policy $\{(s_5, o_3, +read), (s_5, o_3, -execute), (s_2, o_3, +write), (s_1, o_1, +write), (s_1, o_1, -execute), (s_3, o_2, +read), (s_2, o_5, +execute)\}$. We assume that the reader has basic knowledge of Coq syntax and concepts for the upcoming specification fragments.

```
Definition Authorisation_Policy := set Authorisation_Term.
Definition AUTH : Authorisation_Policy :=
 (authorisation (subject 5) (object 3) (Signed_Action plus read)) ::
 (authorisation (subject 5) (object 3) (Signed_Action minus execute)) ::
 (authorisation (subject 2) (object 3) (Signed_Action plus write)) ::
 (authorisation (subject 1) (object 1) (Signed_Action plus write)) ::
 (authorisation (subject 1) (object 1) (Signed_Action minus execute))::
 (authorisation (subject 3) (object 2) (Signed_Action plus read)) ::
 (authorisation (subject 2) (object 5 ) (Signed_Action plus execute)) ::
 nil.
```

Figure 7.14. The example authorization policy specification for conflict checking.

7.6.2. Equality Functions and Decidable Equality

Function definitions on types require Boolean equality functions, whereas proof definitions require equality lemmas. Coq has a well-defined basis for equality of basic pre-defined types such as nat, bool etc. but the proof author needs to define basic lemmas for defining and checking equality of new types. Checking the equality of subjects and objects is based on equality of natural numbers. The *beq_nat* function to check equality of natural numbers is defined in Coq standard library. The Boolean equality functions return true if two given terms are equal and false otherwise. The *eq_as_bool* function checks if two subjects s_i and s_j are equal. Naturally, $s_i = s_j$ if $i = j$. For every boolean equality functions some lemmas should be proved. The lemma *beq_eq_true* defined in the Coq standard library specifies that when a boolean equality function returns true, the input parameters are equal. For authorization terms, the corresponding lemma is *eq_aterm_beq_eq*. The converse of this lemma, *eq_aterm_bool_shows_equal*, states that whenever two authorization terms are equal, boolean equality function returns true. When two terms are different the boolean equality function should return false. This is specified with the lemma *eq_aterm_bool_shows_diff*. Boolean equality functions also need to satisfy the identity equality $a = a$ specified as the lemma *eq_aterm_bool_id*. Since an authorization term contains a subject, an object and a signed action, we have also proved all the lemmas for these types with the same pattern prior to proof of equality lemmas for authorization terms. Decidable equality lemmas are needed for set operations in the standard Coq library on user-defined types. The type *eqdec* defines a decidable equality type. For each user defined type that requires set operations a theorem that matches this pattern must be proved. For authorization terms the theorem *auth_eq_dec* has been proved. Once again proving this theorem requires that decidable equality lemmas be proven on subjects, objects and signed actions. The Coq code is presented in Figure 7.15.

7.6.3. Computation on Authorization Policy

An authorization policy may be specified as a pre-defined set or an empty set. In both cases, add and remove operations should be supported and both operations

```
Definition eq_as_bool (as1 as2: AuthorisationSubject) : bool :=
 match as1 with subject n1⇒
   match as2 with subject n2 ⇒ eq_nat_bool n1 n2 end
end.
Definition beq_eq_true: ∀ x y:A, x =y true = beq x y.
Lemma eq_aterm_beq_eq :
  ∀ x y: Authorisation_Term, eq_aterm_bool x y = true → x = y.
Lemma eq_aterm_bool_shows_equal :
  ∀ aterm1 aterm2: Authorisation_Term,
    aterm1= aterm2 → eq_aterm_bool aterm1 aterm2 = true.
Lemma eq_aterm_bool_shows_diff :
  ∀ aterm1 aterm2: Authorisation_Term,
    aterm1 ≠ aterm2 → eq_aterm_bool aterm1 aterm2 = false.
Lemma eq_aterm_bool_id:
  ∀ aterm : Authorisation_Term, eq_aterm_bool aterm aterm = true.
Definition eqdec (A:Set) := ∀ a b : A, {a = b}+{a ≠ b}.
Theorem auth_eq_dec: eqdec Authorisation_Term.
```

Figure 7.15. Specification of equality functions and decidable equality.

must not introduce modal conflicts. The function *Add_Authorization* adds an authorization term to an existing authorization policy. The function *find_negative_or_same* checks if the given authorization term already exists, and if a negatively signed authorization term for the given term exists in the policy. If not, the given term is appended to the head of the policy list. Otherwise the policy is returned as is. The *find_negative_or_same* function does this check for every given pair of authorization terms and policy. This function is a partial and recursive function. Partial functions return the type *option A*, where *A* is of sort *Set*. In this case *A* = *Authorization_Term*. For values that the function is defined, *Some a:A* is returned; *None* is returned when the function is undefined. *Some a* is returned when the head of the list corresponds to the same or a negatively signed term and *None* is returned when the list is empty, i.e. no term with these properties has been found. The searching continues by recursively calling the function with the remaining part of the policy list.

The remove function *Remove_Authorization* finds and removes an authorization term from the policy and returns the remaining policy. This recursive function works by comparing the head of the policy list with the term to be removed. If the two terms are the same (checked with *eq_aterm_bool*) then the search continues with the head element removed, otherwise, the function searches the remaining list and keeps the head element. The function terminates if there is no more term left to search. The Coq specification of the functions for computation on authorization policy is given in Figure 7.16.

7.6.4. Defining and Checking Conflict-Free Properties

The predicates presented in Figure 7.17 are used for checking conflicts in policies. The first predicate, *No_Conflict* asserts that a given policy is modal conflict-free. *Contains_No_Conflicting* predicate asserts that a new term that is intended to be added to a policy does not cause a modal conflict. These predicates are "inductive predicates". Inductive predicates in Coq are similar to predicates in Prolog, but have higher expressive power. They are defined using constructors, just like an inductive data type, with the difference that they are in the sort Prop rather than Set. The constructors define the propositions similar to Peano induction $P(0), P(n) \rightarrow P(n+1)$, where P is a predicate. The inductive predicates defined below are recursive in the sense that they refer to their own names. The *No_Conflict* predicate states the following:

(i) Any empty policy is conflict free,
(ii) Any policy with only one term is conflict free,
(iii) If two given terms x and y are not mutually negatively signed, and adding x and y separately does not introduce a conflict, then adding both elements does not introduce a conflict.

The predicate *Not_Negative_Sign* asserts that two authorization terms x and y are not mutually negatively signed. This predicate states that the negatively signed form of x should be different from y.

```
Definition Add_Authorisation (new_at: Authorisation_Term)
(auth_policy : Authorisation_Policy) : Authorisation_Policy :=
  match find_negative_or_same auth_policy new_at with
    | None ⇒ new_at :: auth_policy
    | Some y ⇒ auth_policy
  end.
Fixpoint find_negative_or_same (l:Authorisation_Policy)
(elem:Authorisation_Term){struct l}:option Authorisation_Term :=
  match l with
    nil ⇒ None
    | a::tl ⇒ match negative_or_same a elem with
        true ⇒ Some a
     | false ⇒ find_negative_or_same tl elem
     end
  end.
Fixpoint Remove_Authorisation (rem_at: Authorisation_Term)
(auth_policy : Authorisation_Policy) {struct auth_policy} : Authorisation_Policy
:=
    match auth_policy with
    | nil ⇒ auth_policy | a::x ⇒
         match eq_aterm_bool rem_at a with
         | true ⇒ x
         | false ⇒ a :: Remove_Authorisation rem_at x
         end
    end.
```

Figure 7.16. Specification of functions for computation on authorization policy.

In order to verify that a given policy AP is conflict free, a theorem of the form *No_Conflict AP* needs to be proven. This requires successive application of *cons_no_conflict* until the goal reduces to a policy with a single element or no elements, which is then proved by *single_no_conflict* or *empty_no_conflict*. Theorem *AUTH_Conflict_Free* proves that the pre-defined policy AUTH is conflict-free. *NonNegativeAuth* is a custom tactic that unfolds definitions of auxillary functions and applies the discriminate tactic of Coq on equalities of the form $(x = y)$ where x and y are two distinct constructors of the same type. Checking whether a new term would introduce a conflict is achieved as follows:

(i) A new term does not introduce a conflict to an empty policy,

(ii) If the policy contains only one term, the new term should not be mutually negatively signed with the existing one,

(iii) If a new term is not mutually negatively signed with the term at the head of the policy *(x)*, and the remaining policy *(l)* itself would not conflict with the addition of the new term, then the new term would not conflict with the complete policy *(x::l)*.

Verifying that a new element does not introduce a conflict into a policy is achieved in a similar fashion to verifying that a policy is conflict free. The inductive constructors of *Contains_No_Conflicting* are applied until the policy is empty. As an example, the authorization term $(s_4, o_4, +read)$ is checked in the theorem *new_add*. These theorems are presented in Figure 7.17. They are re-usable for verification of other policies and terms.

7.6.5. Verification of Policy Functions

The policy should remain consistent when new authorization terms are added and removed. To ensure consistency, theorems about preservation of the conflict-free property should be proved for the fuctions to add and remove authorization terms. We list the proof scripts for the functions for adding and removing authorization terms, together with the sub-goals generated and the explanation on how they are solved.

```
Inductive No_Conflict : Authorisation_Policy → Prop :=
  | empty_no_conflict : No_Conflict nil
  | single_no_conflict : ∀ a_t: Authorisation_Term, No_Conflict (a_t::nil)
  | cons_no_conflict : ∀ (x y:Authorisation_Term) (l: Authorisation_Policy) ,
    Not_Negative_Sign x y → No_Conflict (cons y l) → No_Conflict (cons x l) →
    No_Conflict (cons x (cons y l)).
Definition Not_Negative_Sign (x: Authorisation_Term) (y:Authorisation_Term):
  Prop:= ¬ (Negative_Signed_Authorisation x = y).
Theorem AUTH_Conflict_Free : No_Conflict AUTH.
  Proof.
    unfold AUTH. repeat apply cons_no_conflict; try NonNegativeAuth;
    apply single_no_conflict.
  Qed.
Ltac NonNegativeAuth :=
  unfold Not_Negative_Sign; unfold Negative_Signed_Authorisation;
  unfold Negative_Signed_Action; unfold Negative_Sign; discriminate.
Inductive Contains_No_Conflicting (aterm: Authorisation_Term) :
Authorisation_Policy → Prop :=
  | empty_not_contain: Contains_No_Conflicting aterm nil
  | single_not_contain: ∀ x: Authorisation_Term,
    Not_Negative_Sign x aterm → Contains_No_Conflicting aterm (x::nil)
  | cons_not_contain : ∀ (x:Authorisation_Term) (l: Authorisation_Policy) ,
    Not_Negative_Sign aterm x → Contains_No_Conflicting aterm l →
        Contains_No_Conflicting aterm (cons x l).
Lemma new_add: Contains_No_Conflicting
  (authorisation (subject 4) (object 4) (Signed_Action plus read)) AUTH.
    unfold AUTH.
    repeat apply cons_not_contain; try NonNegativeAuth.
    apply empty_not_contain.
Qed.
```

Figure 7.17. The predicates and theorems used for checking conflicts in policies.

We will not show the proof of the lemmas. The lines beginning with the > sign present the proof script and the remaining lines are the output of the Coq theorem prover. The first theorem, presented in Figure 7.18, is about the operation for adding an authorization term to a security policy. If a security policy is conflict-free, then adding an element using the *Add_Authorization* function also produces a conflict free policy. The "unfold" tactic replaces the name of the Add_Authorization function with its definition. The Coq output to this tactic lists the hypotheses and new generated sub-goals.

```
Theorem Add_Conflict_Free :
∀ (aterm: Authorisation_Term) (policy: Authorisation_Policy), No_Conflict policy
→ No_Conflict (Add_Authorisation aterm policy).
> intros aterm policy.
> unfold Add_Authorisation.
  aterm : Authorization_Term
  policy : Authorization_Policy
  No_Conflict policy →
  No_Conflict
    match find_negative_or_same policy aterm with
      |Some _=> policy
      |None => aterm :: policy
    end (1/1)
```

Figure 7.18. Theorem for adding an authorization term to a security policy without introducing conflicts.

The "functional induction" tactic derives the proof terms for structurally recursive functions, defines relevant hypothesis for each branch of the function and automatically performs rewriting steps using the equalities generated by the tactic. The application of the functional induction tactic is presented in Figure 7.19. First sub-goal (1/3) is solved by simplification and application of the assumption that a policy with a single term does not have a modal conflict. Second sub-goal (2/3) is solved automatically.

The application of the tactic *generalize*, presented in Figure 7.20, on the third

```
> functional induction first_in_policy_negative_or_same policy aterm.
l : Authorization_Policy
elem : Authorization_Term
H_eq_ : l = nil
No_Conflict nil → No_Conflict (elem :: nil) (1/3)
> intros; simpl; apply single_no_conflict.
No_Conflict (a :: tl) → No_Conflict (a :: tl) (2/3)
> auto.
  No_Conflict (a :: tl) →
  No_Conflict
    match find_negative_or_same tl elem with
      |Some _⇒ a :: tl
      |None ⇒ elem :: a :: tl
    end (3/3)
```

Figure 7.19. The application of the functional induction tactic.

sub-goal *(3/3)* in Figure 7.19 generates 2 new sub-goals. The first sub-goal is readily solvable with the information contained in the goal. The second sub goal requires longer work. We first use the lemma that states removing an element from the head of the policy does not violate the No_Conflict property. This changes our goal to 3 new sub-goals.

This sub-goal states that *elem* and *a* are not mutually negatively signed authorizations. We have this information in hypothesis *H_eq_0* but the symmetrical property of *Not_Negative_Sign* needs to be used and the goal needs to be converted from propositional predicate to boolean predicate. We then unfold the definition of the boolean predicate *negative_or_same* and use the logical rule $P \rightarrow P \vee Q$ (*or_left* in Coq).

The sub-goal (2/3) is already in the hypothesis *Hb*. To prove the sub-goal (3/3), we make use of hypotheses *Ha* and *Hb* to get $No\_Conflict(a :: tl) \rightarrow No\_Conflict(tl)$ which is solvable by a separate lemma named *No_Conflict_Remove* of the same form.

```
> generalize H; case first_in_policy_negative_or_same.
a : Authorization_Term
tl : list Authorization_Term
H_eq_ : l = a :: tl
H_eq_0 : negative_or_same a elem = false
H : No_Conflict tl → No_Conflict
        match find_negative_or_same tl elem with
          |Some _⇒ tl
          |None ⇒ elem :: tl
        end
Authorization_Term  → (No_Conflict tl No_Conflict tl) →
No_Conflict (a :: tl) → No_Conflict (a :: tl) (1/2)
> intros; assumption.
(No_Conflict tl → No_Conflict (elem :: tl)) → No_Conflict (a :: tl) → No_Conflict
(elem :: a :: tl) (2/2)
> intros; apply cons_no_conflict.
Ha : No_Conflict tl → No_Conflict (elem :: tl)
Hb : No_Conflict (a :: tl)
Not_Negative_Sign elem a (1/3)
> apply Not_Negative_Is_Reflexive; apply not_negative_bool_prop.
> unfold negative_or_same in H_eq_0.
> apply or_left with (b:=eq_aterm_bool a elem); assumption.
No_Conflict (a :: tl) (2/3)
> assumption.
No_Conflict (elem :: tl) (3/3)
> apply H0.
> generalize H1; apply No_Conflict_Remove.
```

Figure 7.20. The proof of the theorem about adding authorization terms without introducing conflicts.

This completes the proof of the first theorem.

The second theorem is about the operation for removing an authorization term. The Coq specification of this theorem, together with its proof, is presented in Figure 7.21. For the remove operation, some lemmas will be introduced. These lemmas are based on the Coq function *incl*, which defines set inclusion on lists. The definition *incl* x y states that the list x is included in the list y. The first lemma states that, if a list x is included in another list y, adding the same element z to both lists does not violate this property.

The second lemma states that, if a list (policy) t is included in a list x and x is conflict-free then t is also conflict free. The third lemma states that, a policy from which an authorization term is removed, is always included in the original policy. The second theorem states that if a policy is conflict-free, removing an element using the Remove_Authorization function produces a conflict-free policy.

The first sub-goal is solvable by the auto tactic. The second sub-goal is directly provable with the *No_Conflict_Remove* lemma. For solving the third sub-goal, the inclusion hypotheses presented before are applied in succession, and the proof is completed.

```
Lemma incl_same_add: ∀ (z: Authorisation_Term) (x y: Authorisation_Policy),
  incl x y → incl (z::x) (z::y).
Hypothesis incl_no_conflict: ∀ (x t: Authorisation_Policy),
  incl t x → No_Conflict x → No_Conflict t.
Lemma incl_remove: ∀ (r: Authorisation_Term) (x: Authorisation_Policy),
  incl (Remove_Authorisation r x) x.
Theorem Remove_Conflict_Free :
  ∀ (aterm: Authorisation_Term) (policy: Authorisation_Policy),
  No_Conflict (policy) → No_Conflict (Remove_Authorisation aterm policy).
> intros aterm policy.
> functional induction Remove_Authorisation aterm policy.
rem_at : Authorization_Term
auth_policy : Authorization_Policy
H_eq_: auth_policy = nil
No_Conflict nil → No_Conflict nil (1/3)
> auto.
No_Conflict (a :: x) → No_Conflict x (2/3)
> apply No_Conflict_Remove.
No_Conflict (a :: x) →
 No_Conflict (a :: Remove_Authorization rem_at x) (3/3)
> apply incl_no_conflict with (x:= a::x) (t:= a :: Remove_Authorisation rem_at
x); (t:= a :: Remove_Authorization rem_at x);
  apply incl_same_add;
  apply incl_remove.
> Qed.
```

Figure 7.21. The specification and proof of the theorem about the removal of authorization terms from security policies without introducing conflicts.

8. CASE STUDIES

In this section, two case studies will be presented. The first case study is about a joint research project between a university, a commercial company and a government organization. Within the first case study, we present inter-domain security policies and their formal specifications in the FPFM-RBAC model. We also present an information flow analysis with spatio-temporal model checking using the Ambient Calculus model checker. Information flow analysis includes different kinds of security breach scenarios in multi-domain mobile networks. The focus is on information leakages because of mobility and inter-domain actions. Second, we present an Online Library case study. Online Library case study includes security policy specifications with the XFPM-RBAC policy language.

8.1. Case Study I: Joint Research Project

A university, a commercial company and a government organization are connected via the Internet. The interconnection between the domains are used to exchange information and use specified information resources from each domain to other domains. The overview of the case study is depicted in Figure 8.1.

The university technology development center will be used for inter-domain access by commercial and government organizations. The technology development center hosts joint projects whereby students and researchers from either institution participate as project members. The project members need to access and share information both locally and remotely from different locations. The members of each organization will also be allowed to access the resources and applications through mobile communication and computing devices.

In this section, an inter-domain case study will be presented. The case includes a sample inter-domain security policy with location and mobility constraints. Examples for information flow analysis based on spatial and temporal model checking will be

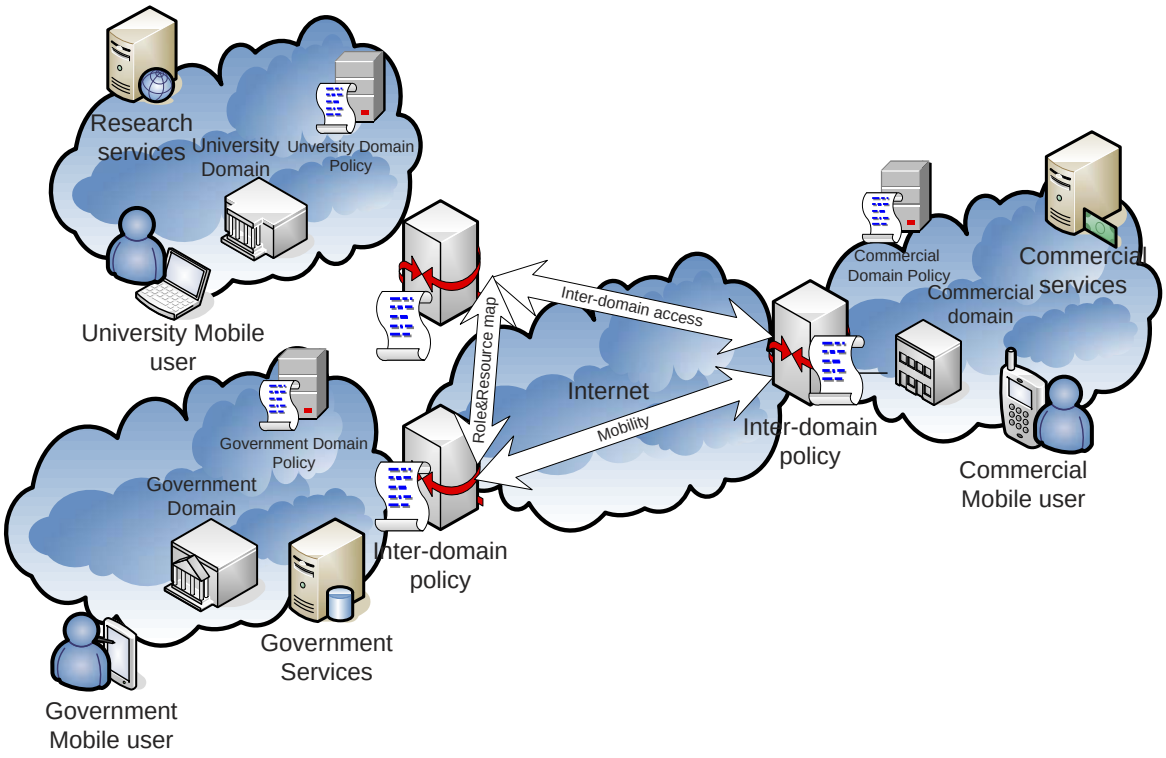

Figure 8.1. Overview of joint research project case study for a university, a commercial company and a government organization.

presented. The analysis focuses on information leakages because of mobility and inter-domain actions.

Our case study scenario is as follows. Three organizations are working on a joint e-health research project. Norman Mullis, who is a professor in Computer Science department, works in the project with Josephine Frantz, who is a researcher in University A, and Cecilia Miele, who is the system administrator for University A. University A hosts a web application server for the project. Commercial company B members Maxine Rundell is the project manager of the consortium and Daniel Mendiola is a software engineer in the project. Florence Mc. Bride, who is a researcher and doctor from the Hospital C, is responsible for assessment of e-health project to medical research. Anthony Weathers, who is a social security expert is responsible for providing research data for social security patients and has the supervisor role in the project.

8.1.1. Inter-Domain Policies for Joint Research Project

The project members have a need to access and share information both locally and remotely from different locations. The members of each organization will also be allowed to access the resources and applications through mobile communication and computing devices.

The main roles and their functions for joint project access are as follows:

- Lecturers: Specific lecturers that work in a joint project as declared and authorised by their respective universities will have access to the joint projects. Clearance level for lecturers is assumed to be "Confidential" i.e. equal to the highest level of classification for shared information. A lecturer from either university will be designated as the project manager for the joint project.
- Research Assistants: Specific research assistants may take responsibility for a joint project. The identities and responsibility of research assistants that are joint project users will be declared and authorised by their respective universities. The access rights will be according to their responsibilities.
- Graduate Students: Graduate students may take part in a joint research project for a specified duration. This time period is during their graduation thesis. A graduate student must be supervised by a lecturer.

The access rights for these roles in the joint project scenario are,

- For lecturers assuming the responsibility of "project manager":
  (i) Add/remove (manage) project users,
  (ii) Add/remove/change (manage) responsibilities,
  (iii) Approve project files,
- For lecturers assuming the responsibility of "supervisor":
  (i) Manage sub-projects, (e.g. for source-code)
  (ii) Create/Read-Write (Edit) files,
  (iii) Create/Edit progress reports,

(iv) Create/Edit Project plans,

(v) Approve sub-project documents.

- For research assistants assuming the responsibility of "system administrator"
  (i) Perform maintenance (e.g) backups,
  (ii) View project users, sub-projects, responsibilities,
  (iii) Perform operations assigned by project manager or supervisor based on delegated access rights. This excludes approval of project and sub-project documents.
- For research assistants/graduate students assuming the responsibility of "researcher"
  (i) Create / Edit files except progress reports and project plans,
  (ii) Read progress reports
  (iii) Read project plans

8.1.2. Formal Specification of Security Policies for Joint Research Project with FPM-RBAC

In this section, we present a case study for the formal specification of security policies with FPFM-RBAC. This case study is based on the Joint Research Project which has been outlined in the previous section. The data sets, role hierarchies, inter-domain access rights, role maps, separation of duty rules, and formalization of inter-domain security policy rules regarding the case study are given.

8.1.2.1. Data Sets of Joint Research Project Case. The mathematical definitions for data sets of University A (UniA), Commercial Company B (CorpB), and Hospital C (HosC) related to the joint project case are given in the specification 8.1 below.

$$\Gamma = \{UniA, CorpB, HosC\}, \tag{8.1}$$
$$H_{UniA} = \{h_{11}, h_{12}, h_{13}\}, H_{CorpB} = \{h_{21}, h_{22}\}, H_{HosC} = \{h_{31}, h_{32}\},$$
$$H = H_{UniA} \cup H_{CorpB} \cup H_{HosC},$$
$$U_{UniA} = \{nmullis, jfrantz, cmiele\},$$

$$
\begin{aligned}
U_{CorpB} &= \{dmendiola, mrundell\},\\
U_{HosC} &= \{fmcbride, aweathers\},\\
U &= U_{UniA} \cup U_{CorpB} \cup U_{HosC},\\
O &= \{jrapp, unif, jrep, prd\}\\
T &= \{Obj, App, File, Db, Portable, Server\}\\
OTH &= \{(jrapp, App), (unif, File), (jrep, File), (prd, Db)\}
\end{aligned}
$$

The elements of the set of hosts represent the following in our scenario: h_{11}: Research lab client of *jfrantz*, h_{12}: Application server used for joint research project of University A, h_{13}: The portable PC of *nmullis*, h_{21}: The portable PC of *mrundell*, h_{22}: The file server of CorpB, which holds joint project reports, h_{31}: File server of Hospital C containing patient records, h_{32}: The portable PC of *fmcbride*. The elements of the set of objects represent the following in our scenario: *jrapp*: Joint research project web application, *unif* : Joint research project work file, *jrep*: Joint research project project report file, *prd*: Patient health records database.

The definitions for inter-domain services are given in the specification 8.2. Here, $\hat{\mathcal{S}}_1$ represents the Joint Research Project inter-domain service of UniA and $\hat{\mathcal{S}}_2$ represents the Patient Health Records inter-domain service of HosC.

$$
\begin{aligned}
\mathcal{I}_{Gamma} =& \{\hat{\mathcal{S}}_1, \hat{\mathcal{S}}_2\} \qquad (8.2)\\
\hat{\mathcal{S}}_1 =& \{\{UniA, CorpB, HosC\}, \{jrapp, unif, jrep\}, \{App, File\},\\
& \{h_{11}, h_{12}, h_{13}, h_{21}, h_{22}, h_{31}, h_{32}\}\},\\
\hat{\mathcal{S}}_2 =& \{\{UniA, HosC\}, \{prd\}, \{App, Db, File\}, \{h_{12}, h_{31}\}\}
\end{aligned}
$$

8.1.2.2. Role Hierarchies of of Joint Research Project Case. In this section we present example role hierarchies for the three domains and the inter-domain role hierarchy related to the joint project scenario. For this purpose the domain role sets and the set

of inter-domain roles are assumed to be as given in the specification 8.3:

$$\begin{aligned} R = R_{UniA} =& \{Member, Teaching, Research, Admin, Lecturer, Faculty, RAssist, \\ & SysAdmin\}, \\ R_{CorpB} =& \{Member, Engineer, Research, Mgr, RnDEng, SwEng, PrjMgr\}, \\ R_{HosC} =& \{Member, Medical, Doctor, SocialSec\}, \\ R_{\Gamma} =& \{JointRes, ResPrj, Admin, ResPrjMgr, ResGrpMgr, Researcher, \\ & Supervisor, SysAdmin\} \end{aligned} \tag{8.3}$$

Note that the structure of role hierarchies is not restricted to a tree, but a tree structure is chosen here for sake of simplicity. The mathematical representations of the role hierarchies for University A RH_{UniA}, B Corporation RH_{CorpB}, Hospital C RH_{HosC} and the inter-domain role hierarchy RH_{Γ} are presented in the specification 8.4.

$$\begin{aligned} RH_{UniA} =& \{Member \prec Teaching, Member \prec Research, Member \prec Admin, \\ & Teaching \prec Lecturer, Research \prec Faculty, Research \prec ResAssist, \\ & Admin \prec SysAdmin\} \\ RH_{CorpB} =& \{Member \prec Engineer, Member \prec Research, Member \prec Mgr, \\ & Engineer \prec SwEng, Research \prec RnDEng, Mgr \prec PrjMgr\} \\ RH_{HosC} =& \{Member \prec Medical, Medical \prec Doctor, Member \prec SocialSec\} \\ RH_{\Gamma} =& \{JointResearch \prec ResPrj, JointResearch \prec Admin, \\ & ResPrj \prec ResPrjMgr, ResPrj \prec ResGrpMgr, ResPrj \prec Researcher, \\ & Admin \prec Supervisor, Admin \prec SysAdmin\} \end{aligned} \tag{8.4}$$

8.1.2.3. Inter-Domain Access Rights for Joint Research Project. In this section we present inter-domain access policy examples for the joint research project case. The "project manager" (*ResPrjMgr*) may use the Joint Project service, but may not access patient health records. In contrast "Supervisor"(*Supervisor*) may access health records but not Joint Project resources. "System administrator" (*SysAdmin*) may perform

administrative duties on Joint Project service, for example to manage and upgrade the joint project web application. A service access matrix for the joint project inter-domain access is given in Table 8.1.

Table 8.1. Service access matrix for joint research project inter-domain access.

Inter-Domain Role	**Inter-Domain Service**	
	Joint Project	*Health Records*
ResPrjMgr	Yes	No
ResGrpMgr	Yes	Yes
SysAdmin	Yes	No
Researcher	Yes	Yes
Supervisor	No	Yes

Detailed access permissions may be granted to these roles based on the services. An excerpt of a permission assignment relation related to the object *jrapp* is given in Table 8.2.

8.1.2.4. Role Maps and Separation of Duty Rules. Here we present some examples for role mapping and separation of duty rules for our case study. We assume the role mapping in the specification 8.5 exists.

$$
\begin{aligned}
RM_h =& \{(Lecturer, ResGrpMgr), (RAssist, Researcher), \\
& (Faculty, Researcher)\} \\
RM_f =& \{(SwEng, Researcher), (RnDEng, Researcher), (Mgr, ResPrjMgr), \\
& (SocialSec, Supervisor), (Medical, Researcher)\} \\
CR_h =& \{RAssist, Faculty\} \\
CR_f =& \{SocialSec, Medical\} \\
CR_\Gamma =& \{Researcher, ResGrpMgr\}
\end{aligned}
\tag{8.5}
$$

In this case, the role assignments $\{(nmullis, Lecturer), (nmullis, Faculty)\}$ would cause a SOD conflict according to $SICR$ since they map to conflicting inter-domain roles. Assignments $\{(fmcbride, Supervisor), (fmcbride, Researcher)\}$ cause a SOD

Table 8.2. A Part of Permission Assignment relation for joint project service relating to object jrapp.

Inter-Domain Role	**Permission**	
	Object	*Action*
ResPrjMgr	jrapp	+execute
ResGrpMgr	jrapp	+execute
SysAdmin	jrapp	+manage, +write
Researcher	jrapp	+execute
Supervisor	jrapp	-execute

conflict according to SRM_f since they are mapped by conflicting foreign roles.

8.1.3. Formal Specification of Inter-Domain Security Policy Rules with FPM-RBAC

Now we present examples for formal specification of inter-domain security policy rules. The formal specification includes the mobility and access control aspects of the inter-domain policy. Example policy statements for mobility and inter-domain access as a verbal form of policy rule are given followed by formal specification of authorization terms and an interpretation of the location and mobility constraints.

(i) "Users with Research Project Manager role from Corporation B are allowed to login with portables to University A.":
(as = ResPrjMgr, ao = Portable, sa =login,
$fo = World[CorpB[T]|p[user[T]|T]|UniA[T]|T]$,
$co = \exists user \in U, EDR(user, UniB) \wedge RAS(user, ResPrjMgr) \wedge OIT(p, Portable))$
Interpretation of the constraints: There is a user, who has assumed the role of a Research Project Manager, which is logged into a host of object type Portable and is not yet connected to either UniA or CorpB, in a location that can be connected to either of the domains UniA and CorpB.

(ii) "Software Engineers of Corporation B can read data from files on the project server in University A domain from within University A.":

(as = SwEng, $ao = h_{12}$, sa = read,
$fo = World[UniA[h_{12}[file[data]|T]| \diamond user[]|T]|CorpB[T]|T]$,
$co = EDR(user, CorpB) \wedge ADU(user, UniA) \wedge RAS(user, SwEng))$
Interpretation of the constraints: UniA contains a project server including a file with data and the user who has assumed the role Software Engineer, is an active domain user of UniA and an enrolled domain user of CorpB, is located somewhere inside UniA.

(iii) "Researchers of University A with a Portable host within Hospital C can not write files to the patient records database in servers of Hospital C.":
(as = Researcher,ao= Server, sa = (-) write,
$fo = World[HosC[p[user[f[]]]|h_{31}[prd[]|T]|T]|T]$,
$co = EDR(user, UniA) \wedge ADU(user, HosC) \wedge RAS(user, Researcher)$
$\wedge OIT(p, Portable) \wedge OIT(prd, Db) \wedge OIT(f, File) \wedge OIT(h_{31}, Server))$
Interpretation of the constraints: The user who has assumed the role Researcher, is an enrolled domain user of UniA, and an active domain user of HosC, who is logged onto a Portable host with a file, is within UniA domain which contains a Server h_{31} with a Database *prd*.

8.1.4. Information Flow Analysis for Inter-Domain Security Policies

Use of the FPM-RBAC policy model enables specification and analysis of information flow policies. The Ambient Calculus model checker presented in our previous work [41] is utilized for this purpose. A novel aspect of FPM-RBAC is that location and mobility of users, objects and hosts may be included in information flow analysis and mobility within multiple domains may be addressed. Here we present two examples, first concerning two domains and the second concerning multiple domains.

8.1.4.1. Information Flow Analysis between Two Domains. In the first example there are two domains, UniA and HosC. Users may roam between domains, but may be logged onto one domain. Assume that a security policy rule related to information flow states that data related to other research projects of University A should not be

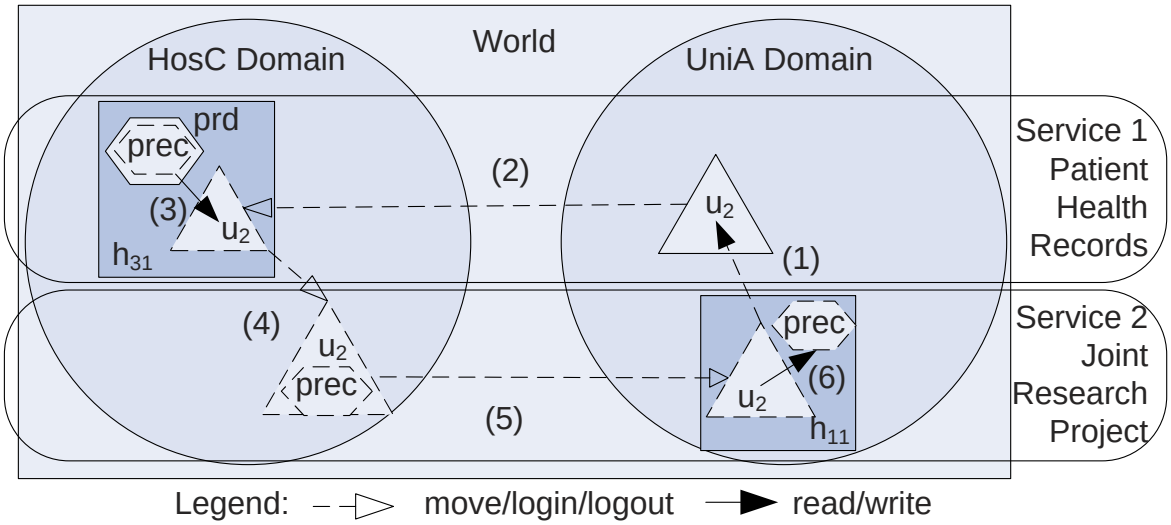

Figure 8.2. An example mobile user inter-domain access leading to an information flow policy violation.

in the same location with patient health records. This is formalized with the mobility constraint (*ILFormula*). Joint researchers are allowed to login to servers and read files and databases within UniA and HosC.

In Figure 8.2, an information leakage breach in this setting is shown. The set of possible actions allowed by the security policy is modeled as an Ambient Calculus specification ($ILSpec1$) and the information flow policy rule is modeled as an Ambient Logic formula ($ILFormula$) in Equation 8.6. In order to find security breaches, the specification includes all possible actions of the user and possible movements of the patient health record, encoded as *pdata* within the database *prec*. Here the users *fmcbride* and *jfrantz* with mobile devices have been encoded respectively as u_1 and u_2.

$$\begin{aligned} ILSpec1 ::= & HosC[u_1[]|h_{31}[prec[pdata[\textbf{in}\ u_2.0|\textbf{out}\ u_2.0]]]]|UniA[h_{11}[u_2[\textbf{out}\ h_{11}.0| \\ & \textbf{out}\ UniA.0|\textbf{in}\ HosC.\textbf{in}\ h_{31}.0|\textbf{out}\ h_{31}.\textbf{out}\ HosC.0|\textbf{in}\ UniA.\textbf{in}\ h_{11}.0| \\ & \textbf{in}\ unif.0|\textbf{in}\ prec.0|\textbf{out}\ prec.0|\textbf{out}\ unif.0]|unif[udata[]]]] \end{aligned} \tag{8.6}$$

$$ILFormula ::= \Box\{\neg\Diamond\{\diamond\{pdata[T]|udata[T]|T\}\} \tag{8.7}$$

The model checker finds the sequence of actions that violate the security policy rule

ILFormula. The action sequence in specification 8.8 formalized by Ambient Calculus capabilities leads to a violation of information flow policy:

$$LCONF_0 \; \overrightarrow{u_2[out\ h_{11}.out\ UniA.in\ HosC.in\ h_{31}]} \\ \overrightarrow{pdata[in\ u_2]}\overrightarrow{u_2[out\ h_{31}.in\ UniA.in\ h_{11}]}\overrightarrow{pdata[out\ u_2]}\; LCONF_1 \quad (8.8)$$

The interpretation of these actions is, the university researcher u_2 (*jfrantz*) moves from *UniA* domain to *HosC* domain with a mobile device, reads patient health records from *HosC* file server to his mobile device, moves it to the university domain and instead of writing it to the joint research application server, writes it to the research lab client where it can be read by other *UniA* users. Although all of the actions in the specification are allowed individually by the security policy, a particular sequence of actions may lead to a security breach as shown in this example.

8.1.4.2. Multi-Domain Information Flow Analysis for Inter-Domain Security Policies. Assume that the there are security policy rules stating that patient records database *prd* may be read directly by users of UniA and HosC but not by CorpB users. The joint project users in UniA may also write records to joint project web application *jrapp* of UniA. Assume that the security policy allows *jrapp* in Host h_{12}, to be accessible and readable by Software Engineers in CorpB (*dmendiola* is encoded as u_4). Also assume that patient records server Host h_{31} in Domain HosC is accessible by Domain UniA user *nmullis* (encoded as u_2). These policy rules result in the formalization of security policy *LCONF* and the mapping of permissions to the formal specification as detailed in Table 8.3.

Now we specify an information flow security policy rule that patient records in HosC domain should not ever occur within CorpB file server h_{22}. This results in a security policy rule with spatial and temporal formula specified as in specification 8.9.

$$fo = \Box\{\neg\diamondsuit\{\diamond\{h_{22}[\diamond\{prec[T]|T\}]\}\}\} \quad (8.9)$$

Table 8.3. Formal specification in Ambient Calculus for multi-domain scenario.

Subject	Permission	Formal Specification LCONF in Ambient Calculus
		$World[HosC[h_{31}[$
u_2	$(read, prd)$	$prd[out\ h_{31}.out\ HosC.in\ UniA.in\ h_{11}.in\ u_2.0\|$
u_2	$(write, prec)$	$prec[in\ jrapp.0]]]]\|$
u_4	$(login, jrapp)$	$UniA[h_{12}[jrapp[out\ h_{12}.out\ UniA.in\ CorpB.in\ h_{22}.in\ u_4.0$
u_2	$(login, jrapp)$	$\|out\ h_{12}.in\ h_{11}.in\ u_2.0$
u_2	$(write, jrapp)$	$\|out\ u_2.out\ h_{11}.in\ h_{12}.0]]$
u_2	$(read, prd)$	$\|h_{11}[u_2[open\ prd.0]]]\|$
u_4	$(read, jrapp)$	$DomainC[h_{22}[u_4[open\ jrapp.0]]]]$

The specifications are given to the model checker to find a counter-example which satisfies $LCONF \models fo$.

The model checker generates a total of 81782 states for this scenario. The flow of states leading to a state in which *fo* is invalidated denotes the sequence of actions leading to the unintended information flow. The sequence of actions, corresponding to the states leading to a counterexample discovered by the model checker is outlined in specification 8.10.

$$LCONF_0 \xrightarrow{u_2(read, prd)} LCONF_1 \xrightarrow{u_2(login, jrapp)} \xrightarrow{u_2(write, prec)} LCONF_2 \xrightarrow{u_4(login, jrapp)} LCONF_3 \xrightarrow{u_4(read, jrapp)} LCONF_4 \quad (8.10)$$

According to this sequence of actions, User nmullis (u_2), who is logged into h_{11} in UniA, reads patient records *prec* from patient records database *prd* in h_{31} ($LCONF_0 \rightarrow^* LCONF_1$). u_2 logs into application *jrapp* and writes *prec* to application *jrapp* in h_{12} ($LCONF_1 \rightarrow^* LCONF_2$). Then software engineer dmendiola (u_4) of CorpB logs into *jrapp* ($LCONF_2 \rightarrow^* LCONF_3$). Then u_4 reads and discloses the database records from within file server h_{22} in CorpB ($LCONF_3 \rightarrow^* LCONF_4$). The state $LCONF_4$ violates the policy rule for information flow.

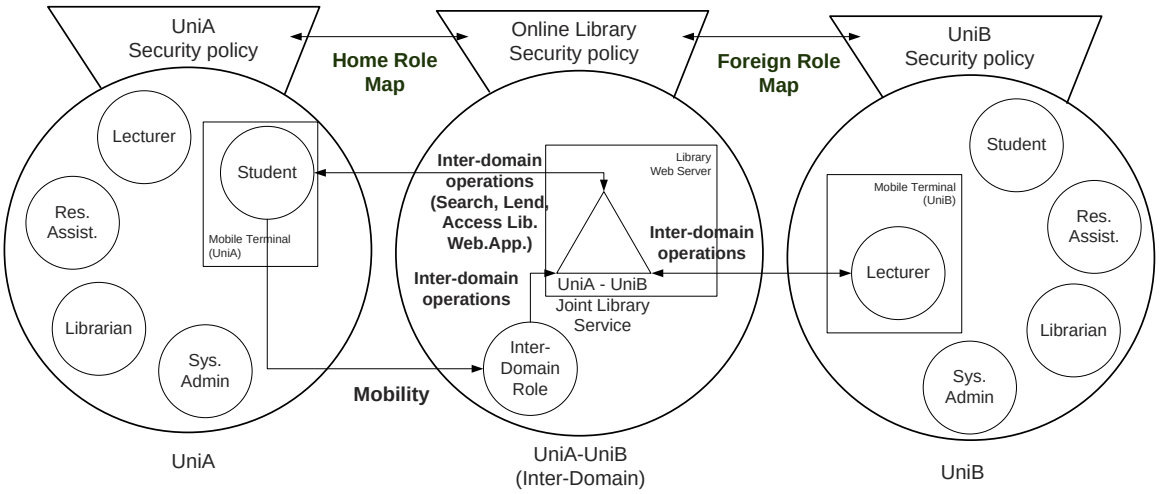

Figure 8.3. An online library with inter-domain access between two universities.

8.2. Case Study II: Online Library

As a case study for policy specification in XFPM-RBAC, we assume that two universities *UniA* and *UniB* provide common information services for the students, research assistants and lecturers. The users are mobile and can roam in the network between domains, access resources in another domain or the Internet. As a specific inter-domain service, two universities provide on-line library access for on-line and mobile users. The libraries allow lending books and periodicals to users of both institutions. The libraries have an online information system for the borrowers to search, reserve publications and get information on their borrowing status, as well as access online publications subscribed by that library. The overview of inter-domain security policy elements of the online library are depicted in Figure 8.3.

Library information consists of periodicals and online publications of other universities that are accessible by online subscription, information about hard-copy publications that reside in the library and user specific information (such as borrowing status).

Users access the online library from within each university campus, from within each library or by remote access from the Internet. The location of an access has an

effect on the available set of permissions. Library services are provided through a web application. The web application uses a database in the background.

We present XML specifications and results of example XSLT translations defined according to XFPM-RBAC security policy language for the online library case. Since the full specification for the Online Library is quite comprehensive, only the main structure of the specification including some specific examples are presented.

8.2.1. Domain and Inter-Domain Policies for Online Library

The main roles and their access rights for multi-domain access to library are as follows:

- Lecturers will be able to connect to online library system from Internet and use local library computers for browsing. They will be able to use the lending facilities of the library information system.
- Research Assistants will be able to connect to online library system from Internet and use local library computers for catalogue browsing. The restriction access to the lending interface outside the university is not allowed.
- Students will be able to access the electronic information resources but will not be able to use local services from the library of another university.
- Librarians enter information such as online subscriptions, hard-copy publication information and user specific information.
- System administrators define security policy and services for the system.

The mathematical definitions for data sets of $UniA$ and $UniB$ are given in the specification 8.11.

$$\Gamma = \{UniA, UniB\} \tag{8.11}$$
$$H_{UniA} = \{H_{11}, H_{12}, H_{13}, H_{14}, H_{15}\}, H_{UniB} = \{H_{21}, H_{22}, H_{23}, H_{24}, H_{25}\}$$
$$H = H_{UniA} \cup H_{UniB}$$

$$U_{UniA} = \{nmullis, jfrantz, cmiele\}, U_{UniB} = \{mrundell, dmendiola, fmcbride\}$$
$$U = U_{UniA} \cup U_{UniB}$$
$$R_{UniA} = \{Member, Student, Research, ResAssist, Lecturer, Librarian, Admin, SysAdmin\}$$
$$R_{UniB} = R_{UniA}$$

The elements of the set of hosts represent the following in our scenario:

- H_{11}/H_{21}: Library web server of University A/B, for accessing the Library Information System.
- H_{12}/H_{22}: A mobile terminal of a student of University A/B.
- H_{13}/H_{23}: A mobile terminal of a lecturer of University A/B.
- H_{14}/H_{24}: A mobile terminal of a research assistant of University A/B.
- H_{15}/H_{25}: A fixed terminal within the library of University A/B.

As an example service definition, the Library service of *UniA* for our example case consists of the specification 8.12.

$$\mathcal{S}_{Library} = \{\mathcal{D}_1, H_1^{Library}, O_1^{Library}, T_1^{Library}\}, \tag{8.12}$$
$$\mathcal{D}_1 = \{UniA\}, H_1^{Library} = \{H_{11}, H_{15}\}$$
$$O_1^{Library} = \{Books, Periodicals, Lib\_Web\_App, Lib\_Search, Lib\_Lend, Lib\_DB\}$$
$$T_1^{Library} = \{App, Files, Web\_App, Database\}$$

Here, *Lib_App* represents an electronic information resources service access for the library, *Lib_Lend* represents a hard copy lending service access for the library, *Lib_Search* represents a catalogue browsing service access for the library and *Lib_DB* represents a library database. A part of Object Type Hierarchy (OTH) relation concerning the Library service is given in the specification 8.13.

$$OTH \subset \{(App, Web\_App), (Files, Books), (Files, Periodicals), \tag{8.13}$$

$(Web\_App, Lib\_App), (Web\_App, Lib\_Search), (Web\_App, Lib\_Lend),$
$(Database, Lib\_DB)\}$

A tree structure is chosen here for the role hierarchies. As an example, the role hierarchy for *UniA*, RH_{UniA}, is given in specification 8.14. For the sake of simplicity we assume the symmetric role hierarchy also exists in *UniB*.

$$\begin{aligned} RH_{UniA} = \{&Member \prec Research, Member \prec Student, Member \prec Admin, \\ &Research \prec Lecturer, Research \prec ResAssist, Admin \prec Librarian, \\ &Admin \prec SysAdmin\} \end{aligned} \tag{8.14}$$

In the specification 8.15, we assume the following inter-domain roles R_Γ, the inter-domain hierarchy RH_Γ and the home role mapping RM_h. Since we have previously assumed $RH_{UniA} = RH_{UniB}$ for the sake of simplicity, we have $RM_h = RM_f$.

$$\begin{aligned} R_\Gamma &= \{Guest, Guest\_Student, Guest\_Researcher, Guest\_Lecturer\} \\ RH_\Gamma &= \{Guest \prec Guest\_Student, Guest \prec Guest\_Researcher, \\ &\quad Guest\_Researcher \prec Guest\_Lecturer\} \\ RM_h &= \{(Lecturer, Guest\_Lecturer), (Lecturer, Guest\_Researcher), \\ &\quad (ResAssist, Guest\_Researcher), (Student, Guest\_Student)\} \end{aligned} \tag{8.15}$$

8.2.2. Domain Configuration, Inter-Domain Configuration and Multi-Domain Security Policy for Online Library with XPFM-RBAC

The XFPM-RBAC specifications for the online library case are given in Appendix C. In Figure C.1, we present a part of the XML file which defines a domain configuration. The domain configuration includes hosts, users, roles, objects, role hierarchy and object type hierarchy for the *UniA* domain.

A part of the inter-domain configuration is presented in Figure C.2. This con-

figuration corresponds to the RH_f and RM_f defined by the foreign domain, and R_Γ, RH_Γ and RM_h defined by the home domain.

A security policy specification includes the *Services*, *Policy_Rules* and *Location_Formulas* elements. An example security policy rule is presented in Figure C.3. Here, the policy rule specifies an authorization term $(Lecturer, Lib\_App, execute, lf1, EDR(Lecturer, UniA))$ where $lf1$ is a location formula and EDR specifies the "Enrolled Domain User" predicate which means that an user is registered with a domain. $ED(Lecturer, UniA)$ has the meaning "Lecturer is an enrolled domain user of UniA". Location formulas are specified under the element *Location_Formulas* and are referred within the security policy rules with their identifiers.

The *Services* element includes the service definitions within a security policy. An example service definition for *Lib_Service* is shown in Figure C.4. The service specification includes *Service_Name*, the name of the service and *Service_ID*, the service identifier. It also includes authorization objects, namely the hosts, domains, objects and object types associated with the service by the *Service_Host*, *Service_Domain*, *Service_Object_Type* and *Service_Object* elements. In the example, the host *Host21* is associated with the actions *login* and *enrol*, which constitute the action group named *a*. Association of a set of authorization objects with a set of actions restricts the set of permissions which may be defined in the permission assignment relation related to the service.

8.2.3. Location and Mobility Constraints for Online Library

Based on the case study, we give some example multi-domain security policy rules with location and mobility constraints that may be specified using XFPM-RBAC. Specification 8.16 states that, "Lecturers can use the library services from within either university or from the Internet". Specification 8.17 states that, "Research assistants of University B can access library lending access interface in University A library web server (host H_{11}) only from within University A". Specification 8.18 states that, "Guest

students can not use the library application of UniA from within UniB".

$$(\texttt{as} = Lecturer, \texttt{ao} = Lib\_App, \texttt{sa} = +execute, \tag{8.16}$$
$$\texttt{fo} = UniA[\diamond\ as[]] \vee UniB[\diamond\ as[]] \vee Internet[\diamond\ as]),$$
$$\texttt{co} = ADU(as, UniA) \vee ADU(as, UniB))$$
$$(\texttt{as} = Res\_Assist, \texttt{ao} = Lib\_Lend, \texttt{sa} = +execute, \tag{8.17}$$
$$\texttt{fo} = UniA[\diamond\ as[]] | \diamond\ H_{11}[ao[]]), \texttt{co} = EDR(as, UniB) \wedge ADU(as, UniA))$$
$$(\texttt{as} = Guest\_Student, \texttt{ao} = Lib\_App, \texttt{sa} = -execute, \tag{8.18}$$
$$\texttt{fo} = UniA[\diamond\ ao] | UniB[\diamond\ as]), \texttt{co} = ADU(as, UniA) \vee ADU(as, UniB))$$

Here, $ADU(user, domain)$ specifies the "Active Domain User" predicate which means that the user has activated a login session to the domain. The location formula in specification 8.18 is specified in XML format similarly to the example in Figure 5.9 therefore we omit the XML specification in this section. The XSLT transformation for location formulas discussed in Section 5.5.1 is applied to the XML representation of the rule in specification 8.18. An input to this transformation is the active user which is involved in the action. Assuming the rule is applied for the user *mrundell*, the Ambient Logic formal specification 8.19 is obtained. This formal specification is presented to the Ambient Calculus model checker as a satisfaction condition for spatio-temporal model checking.

$$lf1 ::= AG - \{UniA[SW\{Lib\_App[T]|T\}]|UniB[SW\{mrundell[T]|T\}]\}; \tag{8.19}$$

8.2.4. Generation of Formal Specifications for Online Library with XPFM-RBAC

The formal specification for a service produces an Ambient Calculus specification which is checked against the satisfaction condition explained in the previous section. The formal specifications of the Library Service are obtained by application of the XSLT transformations presented in Section 5.5.2 for the selected service *Lib_Service*.

The inputs to this transformation are the the domain configurations and the security policy definitions presented above. The result is the XML specification for an Ambient Calculus process corresponding to the service *Lib_Service*. A part of this specification is presented in Figure C.5 in Appendix C. We omit the syntactic elements of the calculus and include some of the definitions for the ambients that constitute the location configuration.

When the Ambient Calculus XSLT transformation is applied to an Ambient Calculus specification such as the one shown above, the formal specification in (8.20) is obtained.

$$ID_1 ::= UniA[Host11[Lib\_App[]]|nmullis[]]|UniB[Host21[Books[]]|mrundell[]]; \tag{8.20}$$

The Ambient Calculus model checker checks the satisfaction of the Ambient Calculus specification against the Ambient Logic formal specification presented in the previous section. In this case, the formula is satisfied by the location configuration. The satisfaction of the location formula implies that the rule in specification 8.18 is applicable in the current location configuration and user *mrundell* is denied access to execute *Lib_App*.

8.2.5. Separation of Duty Constraints for Online Library with XPFM-RBAC

Here we present some examples for separation of duty rules for our case study. We assume the following set of conflicting roles and conflicting services: $CR_h = \{ResAssist, Lecturer\}, CR_\Gamma = \{Guest\_Student, Guest\_Lecturer\}$. In this case, the role assignments $\{(nmullis, Student), (nmullis, Lecturer)\}$ would cause a SOD conflict according to $SICR$ since they map to conflicting inter-domain roles. The assignments $\{(fmcbride, Guest\_Researcher), (fmcbride, Guest\_Lecturer)\}$ would cause a SOD conflict according to SRM_h since they are mapped by conflicting home roles.

As an example for constraints evaluation, consider the role-based SOD constraints specification in Figure C.6 in Appendix C. Assume that the user *mrundell* is assigned to roles $\{ResAssist, Lecturer, Sys\_Admin, Admin\}$. According the following constraints specification, conflicting role set $CR_{ID\_10} = \{ResAssist, Lecturer\}$ and $CR_{ID\_11} = \{Librarian, Sys\_Admin, Admin\}$. For CR_{ID_10}, the minimum number of conflicting roles which will result in a conflict is $n\_conflicting = 2$, whereas for ID_11, $n\_conflicting = 3$.

The role assignments of *mrundell* results in a SOD conflict for the conflicting role set CR_{ID_10}. The evaluation of the SOD constraints specification using XSLT produces the XML file containing static SOD conflicts in Figure C.7 in Appendix C.

9. CONCLUSIONS AND FUTURE WORK

Multi-domain mobile network environments consist of multiple interconnected domains. Each domain includes mobile users, hosts and objects. Moreover, inter-domain policies in the multi-domain mobile network environment need to support concepts such as locations, mobility, role mapping, inter-domain access rights and separation of duty between domains. We are concerned with formal specification and verification of security policy in an environment where users roam between different administrative domains. The problem here is the formal determination of whether the actions conducted by the users are compliant with the domain and inter-domain security policies, given a model of a network that includes mobile users roaming different administrative domains with their respective domain security policies and an inter-domain security policy.

In this thesis, we have presented the Formal Policy Framework for Mobile Networks (FPFM). The FPFM framework includes methods and tools for specification of mobile network configurations, with their respective security policies and verification of security policies. FPFM is the first security policy framework that includes location and mobility constraints, role mapping, inter domain access rights and separation of duty policy rules for multi-domain security policies. FPFM provides integrated support for verification. This is achieved through a formal security policy model and integrated verification tools. FPFM provides an incremental and automated means of developing formal specification of security policies, makes formal methods usable by security administrators and applications, provides tools to specify and verify complex temporal and locational constraints in policies and integrates various formal specification methods and tools in a single framework.

As the formal basis of the FPFM framework, we have proposed a formal security policy model for multi-domain mobile networks named FPM-RBAC. FPM-RBAC makes use of Ambient Calculus, Ambient Logic and Predicate Logic for specification of security policies. Ambient Logic is used for the specification of dynamic mobility

and location constraints in security policy rules. Ambient Calculus is used for the specification of the current state of a mobile network and breach scenarios for testing of policies. The mobility and location constraints in policy rules are matched to the current state of the mobile network by checking the validity of Ambient Logic formulas against Ambient Calculus specifications. Predicate Logic is used for specification of static constraints within security policy rules, such as domain membership, role assignments or separation of duty.

We have also proposed a XML-based security policy language named XFPM-RBAC. The XFPM-RBAC language builds upon the formal security policy model FPM-RBAC. XFPM-RBAC provides means to access the formal reasoning environment by system administrators, applications and network elements. A Security Policy Management Interface (SPMI) application has been developed for authoring security policies with XFPM-RBAC.

Theorem proving and model checking techniques are used for verification. The Coq proof assistant tool is used for specification of the formal policy model for a given network and for the verification of policies. Each authorization policy is represented in the Calculus of Inductive Constructions (CIC), which is a formal language implemented by the Coq proof assistant. A model checker for Ambient Calculus has been developed as part of the framework which checks the validity of Ambient Logic formulas against Ambient Calculus specifications.

The first contribution of the presented security policy framework is the introduction of a formal inter-domain policy model for mobile networks. Previous studies address the inter-domain policy aspects for stationary environments where the subjects and objects are considered to be known in advance, and within fixed locations. Our inter-domain policy model addresses location and mobility aspects of multi-domain mobile networks. The FPFM framework introduces a distributed administration model, in which inter-domain rules are defined for foreign roles acting on home domain resources. With the help of this administration model, the FPFM framework does not require the global knowledge of users and resources and does not introduce conflicts

between inter-domain rules of different domains.

Another contribution of our framework is the inclusion of a formal mobility model in process calculus within security policy. This model is capable of representing mobile network state as well as complex location and mobility constraints in security policy rules. To the best of our knowledge, our framework is the first example of a security policy framework which includes a formal mobility model. Existing studies which address formal specification of security policy rules with location constraints do not include a formal mobility model. By incorporating a formal process calculus model, we enable formal compliance checking of security policy rules with respect to the state of the mobile network. A novel aspect of the presented model is the support for formal information flow analysis of security policies related to mobility within multiple security domains. To support formal analyses on security policies, we have devised a novel spatio-temporal model checking algorithm for Ambient Calculus with an improved complexity over existing algorithms.

Third contribution is the integration of formal verification tools for model checking and theorem proving into the security policy framework, together with the ability to generate formal specifications from security policies for the purpose of verification. Existing studies on the integration of model checking and theorem proving concentrate on integration based on the tools or the formal languages. We address the integration issue based on the model, where we propose a XML based language that encompasses all the model elements, and generates formal specifications. The approach for generation of formal specifications from XML based specifications can be applied in different settings.

The first area for possible improvement is the enforcement of FPFM security policies. Regarding security policy enforcement, model checking techniques need to be improved, possibly by using on-the-fly model checking approach. Formal methods in the framework should be adopted to efficiently handle incremental changes in security policy and network configuration. An more efficient algorithm for the computation of the authorization function could be provided on this basis. 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 which supports parallel model checking could provide an increase in performance.

Regarding model checking of security policies, the proposed algorithm has proven to be efficient for off-line verification of security policies with rules of reasonable complexity, which is determined by the number of logical connectives. For very complex policies which 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. The number of generated states by the spatial model checker increases exponentially with the number of active ambients. Methods for reducing the size of the state transition system need to be investigated. The size of the state transition system is the most significant element for the time and spatial cost of model checking. *Partial order reduction* techniques decrease the number of the states of the state transition system, and therefore reduce time consumption and size of generated NuSMV code. The aim of partial order reduction is to reduce the number of possible orderings that should be analyzed for checking formula stated in a temporal logic. In partial order reduction, parallel interleavings of action sequences are replaced by a single path fragment which respects the orderings in these sequences. To the best of our knowledge, a partial order reduction for model checking Ambient Calculus specifications does not yet exist. Therefore, a study in this direction could provide an important contribution. Regarding integration approaches for integrating model checking with theorem proving, a study to compare and contrast different approaches could be valuable. In this area there are two different approaches, integration based on the formalism and integration based on the tool. The first approach is to define an integrated formal method, based on process calculus, which follows the deep embedding approach. In this formal language, security policy rules and predicates should be part of the formal language itself. The second approach is to investigate and possibly develop a verification tool which provides integrated model checking and theorem proving. Existing verification tools such as Coq may be utilized for this purpose. The integration of the Ambient Calculus Model Checker with the Coq theorem prover could be investigated.

APPENDIX A: XML SCHEMAS DEFINED IN XFPM-RBAC

```
<xs:complexType name="User_Def">
   <xs:sequence>
      <xs:element name="Assigned_Role"
         type="Role_Assignment" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="User_ID" type="xs:string"
      use="required"/>
   <xs:attribute name="Surname" type="xs:string"
      use="required"/>
   <xs:attribute name="Name" type="xs:string"
      use="required"/>
   <xs:attribute name="Home_Domain_ID" type="xs:IDREF"
      use="required"/>
</xs:complexType>
<xs:complexType name="Host_Def">
   <xs:sequence>
      <xs:element name="Host_Object_ID" type="xs:IDREF"
         maxOccurs="unbounded"/></xs:sequence>
   <xs:attribute name="Type" type="xs:string"
      use="optional"/>
   <xs:attribute name="Name" type="xs:string"/>
   <xs:attribute name="Host_ID" type="xs:ID" use="required"/>
   <xs:attribute name="Enrolled_Domain_ID" type="xs:IDREF"
      use="required"/></xs:complexType>
<xs:complexType name="Role_Def">
   <xs:sequence>
      <xs:element name="Parent_Role" type="xs:IDREF"
         minOccurs="0"/>
      <xs:element name="Role_ID" type="xs:ID"/>
      <xs:element name="Role_Name" type="xs:string"/>
   </xs:sequence></xs:complexType>
<xs:complexType name="Constraint_Def">
   <xs:sequence>
      <xs:element name="Condition" type="xs:string"/>
      <xs:element name="Formula" type="xs:string"/>
   </xs:sequence></xs:complexType>
<xs:element name="Domain_Def">
   <xs:annotation>
      <xs:documentation>Definition of a Domain in
         FPFM</xs:documentation>
   </xs:annotation>
```

Figure A.1. Domain Configuration XML Schema of FPM-RBAC.

```
<xs:complexType>
   <xs:sequence>
      <xs:element name="Hosts">
         <xs:complexType>
            <xs:sequence>
               <xs:element name="Host" type="Host_Def"
                  maxOccurs="unbounded"/>
            </xs:sequence></xs:complexType></xs:element>
      <xs:element name="Users">
         <xs:complexType><xs:sequence>
               <xs:element name="User" type="User_Def"
                  maxOccurs="unbounded"/>
            </xs:sequence></xs:complexType></xs:element>
      <xs:element name="Roles" nillable="1">
         <xs:complexType>
            <xs:sequence>
               <xs:element name="Role" type="Role_Def"
                  maxOccurs="unbounded"/>
            </xs:sequence></xs:complexType>
         <xs:key name="Role_Key">
            <xs:selector xpath="Role"/>
            <xs:field xpath="Role_ID"/>
         </xs:key>
         <xs:keyref name="Parent_Role_Key" refer="Role_Key">
            <xs:selector xpath="Role"/>
            <xs:field xpath="Parent_Role"/>
         </xs:keyref>
         <xs:unique name="Role_Key_Unique">
            <xs:selector xpath="Role"/>
            <xs:field xpath="Role_ID"/>
         </xs:unique>
      </xs:element>
      <xs:element name="Object_Type_Hierarchy">
         <xs:complexType>
            <xs:sequence>
               <xs:element name="Object_Type"
                  type="Object_Type_Def" maxOccurs="unbounded"/>
            </xs:sequence></xs:complexType>
      </xs:element>
      <xs:element name="Objects">
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="Object" type="Object_Def"
                     maxOccurs="unbounded"/>
               </xs:sequence>
            </xs:complexType>
      </xs:element>
```

Figure A.2. Domain Configuration XML Schema of FPM-RBAC (cont.).

```
   <xs:element ref="Role_Hierarchy"/>
      </xs:sequence>
         <xs:attribute name="Domain_ID" type="xs:ID"
            use="required"/>
         <xs:attribute name="Domain_Name" type="xs:string"
            use="required"/>
      </xs:complexType>
   </xs:element>
   <xs:complexType name="Object_Def">
      <xs:sequence>
         <xs:element name="Type_of_Object">
            <xs:complexType>
               <xs:attribute name="object_type" type="xs:IDREF"
                  use="required"/>
            </xs:complexType>
         </xs:element>
         <xs:element name="Object_Name" type="xs:string"/>
         <xs:element name="Object_ID" type="xs:ID"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="Object_Type_Def">
      <xs:sequence>
         <xs:element name="Object_Type_Name" type="xs:string"/>
         <xs:element name="Object_Type_ID" type="xs:ID"/>
      </xs:sequence>
      <xs:attribute name="Parent_Object_Type" type="xs:IDREF"/>
   </xs:complexType>
   <xs:element name="Role_Hierarchy" nillable="1">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="RH_Node"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:complexType name="Subordinate_Role">
      <xs:attribute name="Parent_Role_ID" type="xs:integer"
         use="required"/>
   </xs:complexType>
   <xs:element name="Object_Type_Hierarchy">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="Object_Type_Hierarchy_Node"
               type="Object_Type_Hierarchy_Node"
               maxOccurs="unbounded"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
```

Figure A.3. Domain Configuration XML Schema of FPM-RBAC (cont.).

```
<xs:complexType name="Subordinate_Object_Type">
   <xs:attribute name="Parent_Object_Type_ID" type="xs:int"/>
</xs:complexType>
<xs:complexType name="Role_Assignment">
   <xs:sequence>
      <xs:element name="Constraint" type="Constraint_Def"/>
      <xs:element name="Assigned_Role_ID" type="xs:IDREF"/>
   </xs:sequence>
</xs:complexType>
   <xs:complexType name="Object_Type_Hierarchy_Node">
   <xs:sequence>
      <xs:element name="Parent_Object_Type"
         type="Object_Type_Def"/>
      <xs:element ref="Object_Type"/>
   </xs:sequence>
</xs:complexType>
<xs:element name="Object_Type" type="Object_Type_Def"/>
<xs:element name="RH_Node">
   <xs:complexType>
      <xs:sequence>
         <xs:element ref="RH_Node" minOccurs="0"
            maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="role_id" type="xs:IDREF"
         use="required"/>
   </xs:complexType>
</xs:element>
```

Figure A.4. Domain Configuration XML Schema of FPM-RBAC (cont.).

```
<xs:element name="Interdomain_Def">
      <xs:annotation>
         <xs:documentation>Describes inter-domain hierarchies
            and role maps</xs:documentation></xs:annotation>
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="Role_Hierarchy" minOccurs="0"/>
            <xs:element name="Foreign_Role_Hierarchy">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="Foreign_Role"
                        type="Role_Def" maxOccurs="unbounded"/>
                  </xs:sequence>
               </xs:complexType>
               <xs:key name="ForeignRoleID_key">
                  <xs:selector xpath="Foreign_Role"/>
                  <xs:field xpath="Role_ID"/>
               </xs:key>
               <xs:keyref name="ForeignParentRole_keyref"
                  refer="ForeignRoleID_key">
                  <xs:selector xpath="Foreign_Role"/>
                  <xs:field xpath="Parent_Role"/>
               </xs:keyref>
            </xs:element>
            <xs:element name="Interdomain_Role_Hierarchy">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="Interdomain_Role"
                        type="Role_Def" maxOccurs="unbounded"/>
                  </xs:sequence></xs:complexType>
               <xs:key name="Interdomain_Role_ID_key">
                  <xs:selector xpath="Interdomain_Role"/>
                  <xs:field xpath="Role_ID"/>
               </xs:key>
               <xs:keyref name="Interdomain_keyref"
                  refer="Interdomain_Role_ID_key">
                  <xs:selector xpath="Interdomain_Role"/>
                  <xs:field xpath="Parent_Role"/>
               </xs:keyref>
            </xs:element>
            <xs:element name="Role_Map_Home">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element ref="Role_Map"
                        maxOccurs="unbounded"/>
                  </xs:sequence> </xs:complexType>
            </xs:element>
```

Figure A.5. Outline of XML Schema of an Inter-Domain Configuration.

```
            <xs:element name="Role_Map_Foreign">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element ref="Role_Map"
                        maxOccurs="unbounded"/>
                  </xs:sequence></xs:complexType></xs:element>
      </xs:sequence></xs:complexType>
      <xs:keyref name="Interdomain_Role_ID_keyref"
         refer="Interdomain_Role_ID_key">
         <xs:selector xpath="Role_Map_Home/Role_Map"/>
         <xs:field xpath="Interdomain_Role_ID"/>   </xs:keyref>
      <xs:keyref name="ForeignRoleID_keyref"
         refer="ForeignRoleID_key">
         <xs:selector
            xpath="Role_Map_Foreign/Role_Map/Mapped_Role"/>
         <xs:field xpath="Mapped_Role_ID"/> </xs:keyref>
      <xs:unique name="Role_ID_unique">
         <xs:selector xpath="*/*"/>
         <xs:field xpath="Role_ID"/>   </xs:unique>
      <xs:key name="Home_Role_ID_key">
         <xs:selector xpath="Role_Hierarchy/RH_Node"/>
         <xs:field xpath="@role_id"/> </xs:key>
      <xs:keyref name="Home_Role_Map_keyref"
         refer="Home_Role_ID_key">
         <xs:selector
            xpath="Role_Map_Home/Role_Map/Mapped_Role"/>
         <xs:field xpath="Mapped_Role_ID"/> </xs:keyref>
      <xs:keyref name="Home_Role_ID_keyref"
         refer="Home_Role_ID_key">
         <xs:selector xpath="Role_Hierarchy/RH_Node"/>
         <xs:field xpath="@role_id"/> </xs:keyref> </xs:element>
   <xs:complexType name="Role_MapType">
      <xs:sequence>
         <xs:element name="Interdomain_Role_ID"
            type="xs:IDREF"/>
         <xs:sequence>
            <xs:element name="Mapped_Role"
               maxOccurs="unbounded">
               <xs:complexType><xs:sequence>
                     <xs:element name="Mapped_Role_ID"
                        type="xs:IDREF"/>
                  </xs:sequence></xs:complexType></xs:element>
               </xs:sequence></xs:sequence></xs:complexType>
   <xs:element name="Role_Map" type="Role_MapType">
      <xs:annotation><xs:documentation>Describes a mapping of
         inter-domain role to a set of roles</xs:documentation>
      </xs:annotation></xs:element></xs:schema>
```

Figure A.6. Outline of XML Schema of an Inter-Domain Configuration (cont.).

```
<xs:element name="Security_Policy">
   <xs:annotation>
      <xs:documentation>Describes a multi-domain security
         policy with location and mobility
         constraints.</xs:documentation>
   </xs:annotation>
   <xs:complexType>
      <xs:sequence>
         <xs:element name="Services">
            <xs:complexType>
               <xs:sequence>
                  <xs:element ref="Service"
                     maxOccurs="unbounded"/>
               </xs:sequence>
            </xs:complexType>
            <xs:unique name="ServiceIDUnique">
               <xs:selector xpath="Service"/>
               <xs:field xpath="Service_ID"/>
            </xs:unique>
         </xs:element>
         <xs:element name="Policy_Rules">
            <xs:complexType>
               <xs:sequence>
                  <xs:element ref="Policy_Rule"
                     minOccurs="0" maxOccurs="unbounded"/>
               </xs:sequence>
               <xs:attribute name="is_interdomain"
                  type="xs:boolean"/>
            </xs:complexType>
         </xs:element>
         <xs:element name="Location_Formulas">
            <xs:complexType>
               <xs:sequence>
                  <xs:element ref="Location_Formula"
                     minOccurs="0" maxOccurs="unbounded"/>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
         <xs:element name="SOD_Constraints">
            <xs:annotation>
               <xs:documentation>Specifies SOD constraints
                  for a policy</xs:documentation>
            </xs:annotation>
            <xs:complexType>
               <xs:sequence>
                  <xs:element ref="SOD_Constraint"
                     minOccurs="0" maxOccurs="unbounded"/>
```

Figure A.7. XML Schema for a multi-domain security policy.

```
            <xs:element name="Conflicting_Sets">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="Conflicting_Role_Set"
                        minOccurs="0" maxOccurs="unbounded">
                        <xs:complexType>
                           <xs:sequence>
                              <xs:element
                                 name="Conflicting_Role"
                                 type="xs:IDREF"
                                 maxOccurs="unbounded"/>
                           </xs:sequence>
                           <xs:attribute name="set_id"
                              type="xs:ID" use="required"/>
                           <xs:attribute name="num_roles"
                              type="xs:integer"
                              use="optional"/>
                        </xs:complexType>
                     </xs:element>
                     <xs:element
                        name="Conflicting_Service_Set"
                        minOccurs="0" maxOccurs="unbounded">
                        <xs:complexType>
                           <xs:sequence>
                              <xs:element
                                 name="Conflicting_Service"
                                 maxOccurs="unbounded"/>
                           </xs:sequence>
                           <xs:attribute name="set_id"
                              type="xs:ID" use="required"/>
                           <xs:attribute
                              name="num_services"/>
                        </xs:complexType>
                     </xs:element>
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
         </xs:sequence>
      </xs:complexType>
      <xs:keyref name="crset_cr_keyref" refer="crset_key">
         <xs:selector xpath="SOD_CR"/>
         <xs:field xpath="@conflicting_set_id"/>
      </xs:keyref>
      <xs:keyref name="crset_keyref" refer="crset_key">
         <xs:selector xpath="SOD_Constraint"/>
         <xs:field xpath="@conflicting_set_id"/>
      </xs:keyref>
```

Figure A.8. XML Schema for a multi-domain security policy (cont.).

.

```
            <xs:key name="crset_key">
               <xs:selector
                  xpath="Conflicting_Sets/Conflicting_Role_Set"/>
               <xs:field xpath="@set_id"/>
            </xs:key></xs:element>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="SOD_Constraint" type="SOD_Constraint_Def"
      abstract="1"/>
   <xs:complexType name="SOD_Constraint_Def">
      <xs:attribute name="constraint_id" type="xs:ID"
         use="required"/>
      <xs:attribute name="conflicting_set_id" type="xs:IDREF"
         use="required"/>
      <xs:attribute name="n_conflicting" use="optional"/>
   </xs:complexType>
   <xs:element name="SOD_CR" type="SOD_Constraint_Def"
      substitutionGroup="SOD_Constraint"/>
   <xs:element name="SOD_CS" type="SOD_Constraint_Def"
      substitutionGroup="SOD_Constraint"/>
```

Figure A.9. XML Schema for a multi-domain security policy (cont.).

.

```
<xs:element name="Policy_Rule">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="Authorization_Subject"
            type="xs:string"/>
         <xs:element name="Authorization_Object"
            type="xs:string"/>
         <xs:element name="Rule_Action" type="Action_Def"/>
         <xs:element name="Location_Formula_Name"
            type="xs:IDREF" minOccurs="0"/>
         <xs:element name="Conditions" type="xs:string"
            minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="auth_obj_type"
         type="Authorization_Object_Type"/>
      <xs:attribute name="service_member_ref" type="xs:IDREF"/>
      <xs:attribute name="role_ref" type="xs:string"
         use="required"/>
   </xs:complexType></xs:element>
<xs:simpleType name="Authorization_Object_Type">
   <xs:restriction base="xs:string">
      <xs:enumeration value="HOST"/>
      <xs:enumeration value="DOMAIN"/>
      <xs:enumeration value="OBJECT"/>
      <xs:enumeration value="OBJECTTYPE"/>
   </xs:restriction></xs:simpleType>
<xs:complexType name="Action_Def">
   <xs:sequence>
      <xs:element name="Action_Type" type="Action_Type"/>
   </xs:sequence>
   <xs:attribute name="action_name" type="xs:string"
      use="required"/>
   <xs:attribute name="sign" type="Action_Sign"/>
</xs:complexType>
<xs:simpleType name="Action_Sign">
   <xs:restriction base="xs:string">
      <xs:enumeration value="plus"/>
      <xs:enumeration value="minus"/>
   </xs:restriction></xs:simpleType>
<xs:complexType name="Action_Group_Def">
   <xs:sequence>
      <xs:element name="Action_Group_Name" type="xs:string"/>
      <xs:sequence>
         <xs:element name="Action" type="Action_Def"
            maxOccurs="10"/>
      </xs:sequence></xs:sequence></xs:complexType>
<xs:element name="Action_Group" type="Action_Group_Def"/>
```

Figure A.10. XML Schema for policy rules.

```
<xs:element name="Service">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="Service_Name"/>
         <xs:element name="Service_ID" type="xs:ID"/>
         <xs:element name="Service_Host"
            type="Service_Member_Type" minOccurs="0"
            maxOccurs="unbounded"/>
         <xs:element name="Service_Domain"
            type="Service_Member_Type" minOccurs="0"
            maxOccurs="unbounded"/>
         <xs:element name="Service_Object_Type"
            type="Service_Member_Type" minOccurs="0"
            maxOccurs="unbounded"/>
         <xs:element name="Service_Object"
            type="Service_Member_Type" minOccurs="0"
            maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="is_interdomain" type="xs:boolean"
         use="required">
         <xs:annotation>
            <xs:documentation>true if service is an
               interdomain service</xs:documentation>
         </xs:annotation>
      </xs:attribute>
   </xs:complexType>
   <xs:unique name="MemberIDUnique">
      <xs:selector xpath="*"/>
      <xs:field xpath="@service_member_id"/>
   </xs:unique>
</xs:element>
<xs:complexType name="Service_Host_Type">
   <xs:attribute name="service_host_id" type="xs:IDREF"
      use="required"/>
</xs:complexType>
<xs:complexType name="Service_Member_Type">
   <xs:sequence>
      <xs:element name="Service_Member_Action_Group"
         type="Action_Group_Def">
         <xs:unique name="ActionTypeUnique">
            <xs:selector xpath="Action"/>
            <xs:field xpath="Action_Type"/>
         </xs:unique></xs:element>
   </xs:sequence>
   <xs:attribute name="service_member_id" type="xs:ID"
      use="required"/>
</xs:complexType>
```

Figure A.11. XML Schema for Services.

```
<xs:element name="Constraints">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="SOD_Constraints">
            <xs:annotation>
               <xs:documentation>Specifies SOD constraints for
                  a policy</xs:documentation>
            </xs:annotation>

            <xs:complexType>
               <xs:sequence>
                  <xs:element ref="SOD_Constraint"
                     minOccurs="0" maxOccurs="unbounded"/>
                  <xs:element name="Conflicting_Sets">
                     <xs:complexType>
                        <xs:sequence>
                           <xs:element
                              name="Conflicting_Role_Set"
                              minOccurs="0"
                              maxOccurs="unbounded">
                              <xs:complexType>
                                 <xs:sequence>
                                    <xs:element
                                       name="Conflicting_Role"
                                       maxOccurs="unbounded">
                                       <xs:complexType>
                                          <xs:attribute
                                             name="cr_id"
                                             type="xs:string"
                                             use="required"/>
                                       </xs:complexType>
                                    </xs:element>
                                 </xs:sequence>
                                 <xs:attribute name="set_id"
                                    type="xs:ID"
                                    use="required"/>
                                 <xs:attribute name="num_roles"
                                    type="xs:integer"
                                    use="optional"/>
                              </xs:complexType>
                           </xs:element>
                           <xs:element
                              name="Conflicting_Service_Set"
                              minOccurs="0"
                              maxOccurs="unbounded">
                              <xs:complexType>
                                 <xs:sequence>
```

Figure A.12. Outline of XML Schema for SOD Constraints.

```
         <xs:element name="Conflicting_Service"
            maxOccurs="unbounded">
                        <xs:complexType>
                                    <xs:attribute
                                       name="cs_id"
                                       type="xs:string"
                                       use="required"/>
                                 </xs:complexType>
                              </xs:element>
                              </xs:sequence>
                              <xs:attribute name="set_id"
                                 type="xs:ID"
                                 use="required"/>
                              <xs:attribute
                                 name="num_services"/>
                           </xs:complexType></xs:element>
                     </xs:sequence></xs:complexType>
               </xs:element></xs:sequence>
         </xs:complexType>
            <xs:keyref name="crset_cr_keyref" refer="crset_key">
               <xs:selector xpath="SOD_CR"/>
               <xs:field xpath="@conflicting_set_id"/>
            </xs:keyref>
            <xs:keyref name="crset_keyref" refer="crset_key">
               <xs:selector xpath="SOD_Constraint"/>
               <xs:field xpath="@conflicting_set_id"/>
            </xs:keyref>
            <xs:key name="crset_key">
               <xs:selector
                  xpath="Conflicting_Sets/Conflicting_Role_Set"/>
               <xs:field xpath="@set_id"/>
            </xs:key></xs:element>
      </xs:sequence>
   </xs:complexType>
</xs:element>
<xs:element name="SOD_Constraint" type="SOD_Constraint_Def"
   abstract="1"/>
   <xs:complexType name="SOD_Constraint_Def">
      <xs:attribute name="constraint_id" type="xs:ID"
         use="required"/>
      <xs:attribute name="conflicting_set_id" type="xs:IDREF"
         use="required"/>
      <xs:attribute name="n_conflicting" type="xs:integer"
         use="optional"/>
      <xs:attribute name="is_dynamic" type="xs:boolean"
         use="optional"/>
   </xs:complexType>
```

Figure A.13. Outline of XML Schema for SOD Constraints (cont.).

```
<xs:element name="SOD_CR" type="SOD_Constraint_Def"
   substitutionGroup="SOD_Constraint"/>
<xs:element name="SOD_CS" type="SOD_Constraint_Def"
   substitutionGroup="SOD_Constraint"/>
<xs:element name="SOD_ICR"
   substitutionGroup="SOD_Constraint">
   <xs:complexType>
      <xs:complexContent>
         <xs:extension base="SOD_Constraint_Def">
            <xs:attribute name="mapped_conflicting_set"
               type="xs:IDREF" use="required"/>
            <xs:attribute name="home_foreign" use="optional"
               default="HOME">
               <xs:simpleType>
                  <xs:restriction base="xs:string">
                     <xs:enumeration value="HOME"/>
                     <xs:enumeration value="FOREIGN"/>
                  </xs:restriction>
               </xs:simpleType>
            </xs:attribute>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
</xs:element>
<xs:element name="SOD_LCR"
   substitutionGroup="SOD_Constraint">
   <xs:complexType>
      <xs:complexContent>
         <xs:extension base="SOD_Constraint_Def">
            <xs:attribute name="sfo" type="xs:IDREF"
               use="required"/>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
</xs:element>
```

Figure A.14. Outline of XML Schema for SOD Constraints (cont.).

```
<xs:element name="Or">
      <xs:complexType>
         <xs:complexContent>
            <xs:extension base="Or_Expr_Def">
               <xs:attribute name="token"
                  type="Location_Tokens" use="optional"
                  fixed="+"/>
            </xs:extension></xs:complexContent>
      </xs:complexType></xs:element>
   <xs:element name="Logical_Expr">
      <xs:complexType>
         <xs:complexContent>
            <xs:extension base="Logical_Expr_Def">
               <xs:attribute name="token"
                  type="Location_Tokens" use="optional"
                  fixed="=&gt;"/>
            </xs:extension></xs:complexContent>
      </xs:complexType></xs:element>
   <xs:element name="Unary" type="Unary_Expr_Def"
      abstract="true"/>
   <xs:element name="Composition">
      <xs:complexType>
         <xs:complexContent>
            <xs:extension base="Composition_Expr_Def">
               <xs:attribute name="token"
                  type="Location_Tokens" use="optional"
                  fixed="|"/>
            </xs:extension></xs:complexContent>
      </xs:complexType></xs:element>
   <xs:element name="Basic_Formula"
      type="Basic_Formula_Expr_Def" abstract="true"/>
   <xs:element name="Location_Formula">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="Formal_Expr" type="xs:string"
               minOccurs="0"/>
            <xs:element name="Formula_Name" type="xs:ID"/>
            <xs:element name="Congruence" fixed="::=">
               <xs:simpleType>
                  <xs:restriction base="Location_Tokens">
                     <xs:enumeration value="::="/>
                  </xs:restriction>
               </xs:simpleType></xs:element>
            <xs:element ref="Logical_Expr"/>
            <xs:element name="Formula_End"
               type="Location_Tokens" fixed=";"/>
         </xs:sequence></xs:complexType></xs:element>
```

Figure A.15. XML Schema for Ambient Logic.

```
<xs:simpleType name="Location_Tokens">
      <xs:restriction base="xs:string">
         <xs:enumeration value="EF"/>
         <xs:enumeration value="EX"/>
         <xs:enumeration value="EG"/>
         <xs:enumeration value="EU"/>
         <xs:enumeration value="AF"/>
         <xs:enumeration value="AX"/>
         <xs:enumeration value="AG"/>
         <xs:enumeration value="AU"/>
         <xs:enumeration value="FORALL"/>
         <xs:enumeration value="EXISTS"/>
         <xs:enumeration value="+"/>
         <xs:enumeration value="*"/>
         <xs:enumeration value="-"/>
         <xs:enumeration value="=&gt;"/>
         <xs:enumeration value="=="/>
         <xs:enumeration value="SW"/>
         <xs:enumeration value="EW"/>
         <xs:enumeration value="|"/>
         <xs:enumeration value="0"/>
         <xs:enumeration value="["/>
         <xs:enumeration value="]"/>
         <xs:enumeration value="T"/>
         <xs:enumeration value="F"/>
         <xs:enumeration value="::="/>
         <xs:enumeration value="{"/>
         <xs:enumeration value="}"/>
         <xs:enumeration value=";"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:complexType name="Logical_Expr_Def">
      <xs:sequence maxOccurs="unbounded">
         <xs:element ref="Or"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="Or_Expr_Def">
      <xs:sequence maxOccurs="unbounded">
         <xs:element ref="Composition"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="Composition_Expr_Def">
      <xs:sequence maxOccurs="unbounded">
         <xs:element ref="Unary"/>
      </xs:sequence></xs:complexType>
   <xs:complexType name="Unary_Expr_Def"/>
   <xs:complexType name="Basic_Formula_Expr_Def"/>
```

Figure A.16. XML Schema for Ambient Logic (cont.).

```
<xs:element name="Somewhere" substitutionGroup="Unary">
   <xs:complexType>
      <xs:complexContent>
         <xs:extension base="Unary_Expr_Def">
            <xs:sequence>
               <xs:element ref="Unary"/>
            </xs:sequence>
            <xs:attribute name="token"
               type="Location_Tokens" use="required"
               fixed="SW"/>
         </xs:extension></xs:complexContent>
         </xs:complexType></xs:element>
<xs:element name="Sometime" substitutionGroup="Unary">
   <xs:complexType>
      <xs:complexContent>
         <xs:extension base="Unary_Expr_Def">
            <xs:sequence>
               <xs:element ref="Unary"/>
            </xs:sequence>
            <xs:attribute name="token"
               type="Location_Tokens" use="required"
               fixed="EF"/>
         </xs:extension></xs:complexContent>
   </xs:complexType></xs:element>
<xs:element name="Everywhere" substitutionGroup="Unary">
   <xs:complexType>
      <xs:complexContent>
         <xs:extension base="Unary_Expr_Def">
            <xs:sequence>
               <xs:element ref="Unary"/>
            </xs:sequence>
            <xs:attribute name="token"
               type="Location_Tokens" use="required"
               fixed="EW"/>
         </xs:extension></xs:complexContent>
   </xs:complexType></xs:element>
<xs:element name="Everytime" substitutionGroup="Unary">
   <xs:complexType>
      <xs:complexContent>
         <xs:extension base="Unary_Expr_Def">
            <xs:sequence>
               <xs:element ref="Unary"/></xs:sequence>
            <xs:attribute name="token"
               type="Location_Tokens" use="required"
               fixed="AG"/>
         </xs:extension></xs:complexContent>
   </xs:complexType></xs:element>
```

Figure A.17. XML Schema for Ambient Logic (cont.).

```
<xs:element name="Not_Expr" substitutionGroup="Unary">
      <xs:complexType>
         <xs:complexContent>
            <xs:extension base="Unary_Expr_Def">
               <xs:sequence>
                  <xs:element ref="Unary"/>
               </xs:sequence>
               <xs:attribute name="token"
                  type="Location_Tokens" use="required"
                  fixed="-"/>
            </xs:extension>   </xs:complexContent>
      </xs:complexType></xs:element>
   <xs:element name="Basic_Formula_Expr"
      substitutionGroup="Unary">
      <xs:complexType>
         <xs:complexContent>
            <xs:extension base="Unary_Expr_Def">
               <xs:sequence>
                  <xs:element ref="Basic_Formula"/>
               </xs:sequence></xs:extension>
         </xs:complexContent>/xs:complexType></xs:element>
   <xs:element name="Ambient" type="Ambient_Expr_Def"
      substitutionGroup="Basic_Formula"/>
   <xs:complexType name="Ambient_Expr_Def">
      <xs:complexContent>
         <xs:extension base="Basic_Formula_Expr_Def">
            <xs:sequence>
               <xs:element name="AmbientName" type="xs:string"/>
               <xs:element name="AmbientStart" fixed="[">
                  <xs:simpleType>
                     <xs:restriction base="Location_Tokens">
                        <xs:enumeration value="["/>
                     </xs:restriction> </xs:simpleType>
               </xs:element>
               <xs:element ref="Logical_Expr" minOccurs="0"/>
               <xs:element name="AmbientEnd" fixed="]">
                  <xs:simpleType>
                     <xs:restriction base="Location_Tokens">
                        <xs:enumeration value="]"/>
                     </xs:restriction></xs:simpleType>
               </xs:element></xs:sequence>
            <xs:attribute name="Ambient_Location_Type"
               type="Location_Type"/>
         </xs:extension></xs:complexContent></xs:complexType>
         <xs:element name="Bracket" type="Bracket_Expr_Def"
            substitutionGroup="Basic_Formula"/>
            <xs:complexType name="Bracket_Expr_Def">
```

Figure A.18. XML Schema for Ambient Logic (cont.).

```
<xs:complexContent>
   <xs:extension base="Basic_Formula_Expr_Def">
   <xs:sequence>
      <xs:element name="BracketStart" fixed="{">
         <xs:simpleType>
            <xs:restriction base="Location_Tokens">
               <xs:enumeration value="{"/></xs:restriction>
            </xs:simpleType></xs:element>
      <xs:element ref="Logical_Expr"/>
      <xs:element name="BracketEnd" fixed="}">
         <xs:simpleType>
            <xs:restriction base="Location_Tokens">
               <xs:enumeration value="}"/>
            </xs:restriction></xs:simpleType></xs:element>
   </xs:sequence></xs:extension></xs:complexContent>
</xs:complexType>
<xs:element name="Nil_Expr" type="Nil_Expr_Def"
   substitutionGroup="Basic_Formula"/>
<xs:complexType name="Nil_Expr_Def">
   <xs:complexContent>
      <xs:extension base="Basic_Formula_Expr_Def">
         <xs:sequence><xs:element name="Nil" fixed="0">
            <xs:simpleType><xs:restriction
               base="Location_Tokens"><xs:enumeration
               value="0"/></xs:restriction></xs:simpleType>
               </xs:element></xs:sequence></xs:extension>
   </xs:complexContent></xs:complexType>
<xs:element name="True_Expr" type="True_Expr_Def"
   substitutionGroup="Basic_Formula"/>
   <xs:complexType name="True_Expr_Def">
      <xs:complexContent>
         <xs:extension base="Basic_Formula_Expr_Def">
            <xs:sequence><xs:element name="True" fixed="T">
                  <xs:simpleType>
                     <xs:restriction base="Location_Tokens">
                        <xs:enumeration value="T"/>
                     </xs:restriction></xs:simpleType>
               </xs:element></xs:sequence></xs:extension>
               </xs:complexContent></xs:complexType>
   <xs:simpleType name="Location_Type">
      <xs:restriction base="xs:string">
         <xs:enumeration value="DOMAIN"/>
         <xs:enumeration value="HOST"/>
         <xs:enumeration value="OBJECT"/>
         <xs:enumeration value="USER"/>
         <xs:enumeration value="WORLD"/>
      </xs:restriction></xs:simpleType>
```

Figure A.19. XML Schema for Ambient Logic (cont.).

```
<xs:element name="Ambient_Calculus_Spec">
<xs:annotation>
   <xs:documentation>Ambient Calculus Specification in
      FPFM</xs:documentation>
</xs:annotation>
<xs:complexType>
   <xs:sequence>
      <xs:element ref="Compilation_Unit"/>
      <xs:element name="Variables">
         <xs:complexType><xs:sequence>
            <xs:element ref="Identifier" maxOccurs="unbounded"/>
            </xs:sequence></xs:complexType>
      </xs:element></xs:sequence>
   <xs:attribute name="Ambient_Calculus_Spec_Name"
      type="xs:string" use="required"/>
</xs:complexType>
<xs:keyref name="Id_Ref_Ambient" refer="Id_Key">
   <xs:selector xpath="Compilation_Unit / Specification /
      Composition / Sequence / Ambient_Expr"/>
   <xs:field xpath="Ambient_Name"/></xs:keyref>
<xs:keyref name="Id_Ref" refer="Id_Key">
   <xs:selector xpath="Compilation_Unit / Specification /
      Composition / Sequence / Path / Action"/>
   <xs:field xpath="Action_Target"/></xs:keyref>
<xs:key name="Id_Key">
   <xs:selector xpath="Variables/Identifier"/>
   <xs:field xpath="@id"/></xs:key></xs:element>
<xs:element name="Compilation_Unit">
<xs:complexType>
   <xs:sequence>
      <xs:element ref="Specification"/>
      <xs:element name="End_of_Spec" type="ENDOFSPEC"/>
   </xs:sequence>
   <xs:attribute name="Compilation_Unit_Name" type="xs:string"
      use="required"/></xs:complexType></xs:element>
<xs:restriction base="xs:string">
   <xs:enumeration value=";"/><xs:enumeration value="0"/>
   <xs:enumeration value="::="/><xs:enumeration value="|"/>
   <xs:enumeration value="["/><xs:enumeration value="]"/>
   <xs:enumeration value="."/><xs:enumeration value="in"/>
   <xs:enumeration value="out"/><xs:enumeration value="open"/>
   <xs:enumeration value="acid"/><xs:enumeration value="mv_in"/>
   <xs:enumeration value="mv_out"/><xs:enumeration value="new"/>
   <xs:enumeration value="("/><xs:enumeration value=")"/>
   <xs:enumeration value="&lt;"/><xs:enumeration value="&gt;"/>
   <xs:enumeration value="{"/><xs:enumeration value="}"/>
</xs:restriction>
```

Figure A.20. XML Schema for Ambient Calculus.

```
<xs:simpleType name="Ambient_Tokens">
</xs:simpleType>
<xs:simpleType name="ENDOFSPEC">
   <xs:restriction base="Ambient_Tokens">
      <xs:enumeration value=";"/>
   </xs:restriction>
</xs:simpleType>
<xs:simpleType name="INACTIVITY">
   <xs:restriction base="Ambient_Tokens">
      <xs:enumeration value="0"/>
   </xs:restriction>
</xs:simpleType>
<xs:simpleType name="CONGRUENCE">
   <xs:restriction base="Ambient_Tokens">
      <xs:enumeration value="::="/>
   </xs:restriction>
</xs:simpleType>
<xs:simpleType name="COMPOSITION">
   <xs:restriction base="Ambient_Tokens">
      <xs:enumeration value="|"/>
      </xs:restriction></xs:simpleType>
<xs:simpleType name="AMBIENTSTART">
   <xs:restriction base="Ambient_Tokens">
      <xs:enumeration value="["/>
   </xs:restriction></xs:simpleType>
<xs:simpleType name="AMBIENTEND">
   <xs:restriction base="Ambient_Tokens">
      <xs:enumeration value="]"/>
   </xs:restriction></xs:simpleType>
<xs:simpleType name="ACTION">
   <xs:restriction base="Ambient_Tokens">
      <xs:enumeration value="."/>
   </xs:restriction></xs:simpleType>
<xs:simpleType name="ENTRANCE">
   <xs:restriction base="Ambient_Tokens">
      <xs:enumeration value="in"/>
   </xs:restriction></xs:simpleType>
<xs:simpleType name="EXIT">
   <xs:restriction base="Ambient_Tokens">
      <xs:enumeration value="out"/>
   </xs:restriction>
</xs:simpleType>
<xs:simpleType name="OBJECTIVE_ENTRANCE">
   <xs:restriction base="Ambient_Tokens">
      <xs:enumeration value="mv_in"/>
   </xs:restriction>
</xs:simpleType>
```

Figure A.21. XML Schema for Ambient Calculus(cont.).

```
<xs:simpleType name="OBJECTIVE_EXIT">
   <xs:restriction base="Ambient_Tokens">
         <xs:enumeration value="mv_out"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:simpleType name="ACID">
      <xs:restriction base="Ambient_Tokens">
         <xs:enumeration value="acid"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:simpleType name="DISSOLUTION">
      <xs:restriction base="Ambient_Tokens">
         <xs:enumeration value="open"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:simpleType name="INPUTSTART">
      <xs:restriction base="Ambient_Tokens">
         <xs:enumeration value="("/>
      </xs:restriction>
   </xs:simpleType>
   <xs:simpleType name="INPUTEND">
      <xs:restriction base="Ambient_Tokens">
         <xs:enumeration value=")"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:simpleType name="OUTPUTSTART">
      <xs:restriction base="Ambient_Tokens">
         <xs:enumeration value="&lt;"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:simpleType name="OUTPUTEND">
      <xs:restriction base="Ambient_Tokens">
         <xs:enumeration value="&gt;"/>
      </xs:restriction>
   </xs:simpleType>
   <xs:simpleType name="BLOCKSTART">
      <xs:restriction base="Ambient_Tokens">
         <xs:enumeration value="{"/>
      </xs:restriction>
   </xs:simpleType>
<xs:simpleType name="BLOCKEND">
      <xs:restriction base="Ambient_Tokens">
         <xs:enumeration value="}"/>
      </xs:restriction>
   </xs:simpleType>

<xs:complexType name="Specification_Type"/>
```

Figure A.22. XML Schema for Ambient Calculus(cont.).

```
<xs:element name="Specification">
   <xs:complexType><xs:sequence>
         <xs:element name="Specification_Name"
            type="xs:string"/>
         <xs:element name="Congruence" type="CONGRUENCE"/>
         <xs:element ref="Composition"/>
      </xs:sequence></xs:complexType></xs:element>
<xs:element name="Composition">
   <xs:complexType><xs:sequence>
         <xs:element ref="Sequence" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="token" type="COMPOSITION"
         use="required"/>
   </xs:complexType></xs:element>
<xs:element name="Sequence">
   <xs:complexType><xs:sequence>
         <xs:element ref="Path"/>
         <xs:element ref="Basic_Expr"/>
      </xs:sequence></xs:complexType></xs:element>
<xs:element name="Path">
   <xs:complexType>
      <xs:sequence minOccurs="0" maxOccurs="unbounded">
         <xs:element ref="Action"/>
      </xs:sequence>
      <xs:attribute name="token" use="optional">
         <xs:simpleType>
            <xs:restriction base="Ambient_Tokens">
               <xs:enumeration value="."/>
            </xs:restriction></xs:simpleType>
      </xs:attribute></xs:complexType></xs:element>
   <xs:element name="Inactivity_Expr"
      substitutionGroup="Basic_Expr">
      <xs:complexType><xs:sequence>
            <xs:element name="Inactivity" type="INACTIVITY"/>
         </xs:sequence></xs:complexType></xs:element>
   <xs:element name="Ambient_Expr"
      substitutionGroup="Basic_Expr">
      <xs:complexType><xs:sequence>
            <xs:element name="Ambient_Name" type="xs:string"/>
            <xs:element name="Ambient_Start"
               type="AMBIENTSTART"/>
            <xs:element ref="Composition" minOccurs="0"/>
            <xs:element name="Ambient_End" type="AMBIENTEND"/>
         </xs:sequence>
         <xs:attribute name="Ambient_Type"
            type="Ambient_Types"/></xs:complexType>
            </xs:element>
```

Figure A.23. XML Schema for Ambient Calculus(cont.).

```
<xs:element name="Basic_Expr" abstract="true"/>
   <xs:element name="Block_Expr" substitutionGroup="Basic_Expr">
      <xs:complexType><xs:sequence>
            <xs:element name="Block_Start" type="BLOCKSTART"/>
            <xs:element ref="Composition"/>
            <xs:element name="Block_End" type="BLOCKEND"/>
         </xs:sequence></xs:complexType></xs:element>
   <xs:element name="Action" type="Action_Type"
      abstract="false"/>
   <xs:element name="Identifier"><xs:complexType>
         <xs:attribute name="name" type="xs:string"
            use="optional"/>
         <xs:attribute name="id_type" type="Identifier_Types"
            use="required"/>
         <xs:attribute name="id" type="xs:ID" use="required"/>
      </xs:complexType></xs:element>
   <xs:simpleType name="Ambient_Types">
      <xs:restriction base="xs:string">
         <xs:enumeration value="DOMAIN"/>
         <xs:enumeration value="USER"/>
         <xs:enumeration value="OBJECT"/>
         <xs:enumeration value="HOST"/>
      </xs:restriction></xs:simpleType>
   <xs:complexType name="Action_Type" abstract="false">
      <xs:sequence><xs:element name="Action_Target"
         type="xs:IDREF"/>
      </xs:sequence><xs:attribute name="action" type="Actions"
         use="required"/></xs:complexType>
   <xs:simpleType name="Actions">
      <xs:restriction base="xs:string">
         <xs:enumeration value="in"/>
         <xs:enumeration value="out"/>
         <xs:enumeration value="mv_in"/>
         <xs:enumeration value="mv_out"/>
         <xs:enumeration value="acid"/>
         <xs:enumeration value="open"/>
         <xs:enumeration value="input"/>
         <xs:enumeration value="output"/>
      </xs:restriction></xs:simpleType>
   <xs:simpleType name="Identifier_Types">
      <xs:restriction base="xs:string">
         <xs:enumeration value="DOMAIN"/>
         <xs:enumeration value="HOST"/>
         <xs:enumeration value="USER"/>
         <xs:enumeration value="OBJECT"/>
         <xs:enumeration value="VARIABLE"/>
      </xs:restriction></xs:simpleType>
```

Figure A.24. XML Schema for Ambient Calculus(cont.).

APPENDIX B: AMBIENT CALCULUS SPECIFICATIONS AND AMBIENT LOGIC FORMULAS FOR PERFORMANCE ANALYSIS OF THE MODEL CHECKING ALGORITHM

```
Spec1::=  World[DomainA[Host1[User1[] |File1[data1[out File1.0|out
          Host1.0|out DomainA.0| in DomainB.0|in DomainC.0|in
          Host4.0 | in Host2.0|out Host2.0|in File3.0|in
          User2.0|out User2.0|in User4.0|out User4.0|in Host3.0
          ]]]] | DomainB[Host3[File3[]]|Host2[User2[in File1.0|in
          File2.0|out File1.0|out File2.0|in File3.0|in File4.0|out
          File3.0|out File4.0]|File2[]]]|DomainC[Host4 [User4[out
          DomainC.0|in DomainB.0| in Host3.0| in File3.0|out
          File3.0|out Host3.0|out DomainB.0|in DomainC.in Host4.
          in File4.0]|File4[]]]]
Spec2::=  World[DomainA[Host1[User1[out DomainA.0 | in
          DomainA.0|in DomainB.0|out DomainB.0|in File1.0|out
          File1.0]|File1[data1[out File1.0|out Host1.0|out
          DomainA.0| in DomainB.0|in DomainC.0|in Host4.0 | in
          Host2.0|out Host2.0|in File3.0|in User1.0|in User2.0|out
          User2.0|in User4.0|out User4.0|in Host3.0]]]] | DomainB
          [Host3[File3[]]|Host2[User2[in File1.0|in File2.0|out
          File1.0|out File2.0|in File3.0|in File4.0|out File3.0|out
          File4.0]|File2[]]]|DomainC[Host4 [User4[out DomainC.0|in
          DomainB.0| in Host3.0| in File3.0|out File3.0|out
          Host3.0|out DomainB.0|in DomainC.in Host4.  in
          File4.0]|File4[]]]]
Spec3::=  World[DomainA[Host1[User1[out Host1.0|out DomainA.0|in
          DomainB.0]|File1[data1[out File1.0|out Host1.0|out
          DomainA.0| in DomainB.0|in DomainC.0|in Host4.0 | in
          Host2.0|out Host2.0|in File3.0|in User1.0|out User1.0|in
          User2.0|out User2.0|in User4.0|out User4.0|in Host3.0]]]]
          | DomainB[Host3[File3[]]|Host2[User2[in File1.0|in
          File2.0 | out File1.0|out File2.0|in File3.0|in
          File4.0|out File3.0|out File4.0]|File2[]]]|DomainC[Host4
          [User4[out DomainC.0|in DomainB.0| in Host3.0| in
          File3.0|out File3.0|out Host3.0|out DomainB.0|in
          DomainC.in Host4.  in File4.0]|File4[]]]]
```

```
Formula1::=  □ { ¬ ♢ { ◇ { Host4[ ◇ { data1[T] | T}] } } }
Formula2::=  □ { World [ DomainA [Host1 [T] | T] | DomainB[Host2 [T]
             | Host3 [T] | T] | DomainC[ Host4 [T] | T]] }
             ∨ ♢{◇{Host4[ ◇{data1[T]}] | T}}
```

APPENDIX C: XFPM-RBAC SPECIFICATIONS FOR THE ONLINE LIBRARY CASE

```
<Domain_Def Domain_ID="Domain1" Domain_Name="UniA"...>
   <Hosts>
      <Host Enrolled_Domain_ID="Domain1"
        Host_ID="Host11" Name="Library_Web_Server">
         <Host_Object_ID>Lib_App</Host_Object_ID> % Other
            objects
      </Host> % Other hosts
   </Hosts>
   <Users>
      <User Home_Domain_ID="Domain1" Name="Norman"
       Surname="Mullis" User_ID="nmullis">
         <Assigned_Role>
            <Constraint>
               <Condition>EDR(nmullis,UniA)</Condition>
               <Formula>locspec2</Formula></Constraint>
            <Assigned_Role_ID>Role2</Assigned_Role_ID>
      </User> % Other users
   </Users>
   <Role_Hierarchy>
      <Role>
         <Parent_Role>Role1</Parent_Role>
         <Role_ID>Role1</Role_ID>
         <Role_Name>Member</Role_Name></Role>
      <Role>
         <Parent_Role>Role1</Parent_Role>
         <Role_ID>Role2</Role_ID>
         <Role_Name>Lecturer</Role_Name>
      </Role> % Other roles
   </Role_Hierarchy>
   <Object_Type_Hierarchy>
         <Object_Type Parent_Object_Type="Object">
               <Object_Type_Name>Applications</Object_Type_Name>
               <Object_Type_ID>App
               </Object_Type_ID>
         </Object_Type> % Other object types
   </Object_Type_Hierarchy>
   <Objects>
      <Object>
         <Type_of_Object object_type="Application"/>
         <Object_Name>Lib_App</Object_Name>
         <Object_ID>Lib_App</Object_ID>
      </Object> % Other objects
   </Objects>
```

Figure C.1. Part of the domain configuration for UniA.

```
<Interdomain_Def ...>
<Foreign_Role_Hierarchy>
   <Foreign_Role>
      <Parent_Role>FRole4</Parent_Role>
      <Role_ID>FRole5</Role_ID>
      <Role_Name>ResAssist</Role_Name>
   </Foreign_Role> % other foreign roles
</Foreign_Role_Hierarchy>
<Interdomain_Role_Hierarchy>
   <Interdomain_Role>
      <Parent_Role>IDRole1</Parent_Role>
      <Role_ID>IDRole2</Role_ID>
      <Role_Name>Guest_Researcher</Role_Name>
   </Interdomain_Role>% other interdomain roles
</Interdomain_Role_Hierarchy>
<Role_Map_Home>
   <Role_Map>
      <Interdomain_Role_ID>IDRole2</Interdomain_Role_ID>
      <Mapped_Role>
         <Mapped_Role_ID>Role2</Mapped_Role_ID>
      </Mapped_Role> % other mapped roles
   </Role_Map> % other role maps
</Role_Map_Home>
<Role_Map_Foreign>
   <Role_Map>
      <Interdomain_Role_ID>IDRole2</Interdomain_Role_ID>
      <Mapped_Role>
         <Mapped_Role_ID>FRole5</Mapped_Role_ID>
      </Mapped_Role> % other mapped roles
   </Role_Map> % other role maps
</Role_Map_Foreign></Interdomain_Def>
```

Figure C.2. Part of the inter-domain configuration.

```
<Policy_Rules>
   <Policy_Rule role_ref="Role2">
      <Authorization_Subject>Lecturer</Authorization_Subject>
      <Authorization_Object>Lib_App</Authorization_Object>
      <Rule_Action action_name="execute">
         <Action_Type>execute</Action_Type>
      </Rule_Action>
      <Location_Formula_Name>lf1</Location_Formula_Name>
      <Conditions>EDR(Lecturer,UniA)</Conditions>
      </Policy_Rule></Policy_Rules>
```

Figure C.3. Inter-domain security policy rule example.

```
<Services>
      <Service is_interdomain="true">
         <Service_Name>Lib_Service</Service_Name>
         <Service_ID>Service1</Service_ID>
         <Service_Host srv_m_id="Host21">
            <Service_Member_Action_Group>
               <Action_Group_Name>a</Action_Group_Name>
               <Action action_name="login">
                  <Action_Type>login</Action_Type>
               </Action>
               <Action action_name="enroll">
                  <Action_Type>enrol</Action_Type>
               </Action></Service_Member_Action_Group>
         </Service_Host> % other service hosts
         <Service_Domain srv_m_id="UniA">
            <Service_Member_Action_Group>
               <Action_Group_Name>b</Action_Group_Name>
               <Action action_name="enrol">
                  <Action_Type>enrol</Action_Type>
               </Action></Service_Member_Action_Group>
         </Service_Domain> % other service domains
         <Service_Object_Type srv_m_id="Web_App">
            <Service_Member_Action_Group>
               <Action_Group_Name>c</Action_Group_Name>
               <Action action_name="execute">
                  <Action_Type>execute</Action_Type>
               </Action></Service_Member_Action_Group>
         </Service_Object_Type> % serv. obj. types
         <Service_Object srv_m_id="Lib_App">
            <Service_Member_Action_Group>
               <Action_Group_Name>d</Action_Group_Name>
               <Action action_name="execute">
                  <Action_Type>read</Action_Type>
               </Action>
               <Action action_name="read">
                  <Action_Type>read</Action_Type>
               </Action></Service_Member_Action_Group>
         </Service_Object> % other service objects
      </Service></Services>
```

Figure C.4. Part of the inter-domain security policy definition for Library service.

```
<Ambient_Calculus_Spec ...>
   <Compilation_Unit
   Compilation_Unit_Name="Lib_Service">
      <Specification>
         <Specification_Name>AC_Lib_Service
         </Specification_Name>
         <Composition token="|">
               <Ambient_Expr Ambient_Type="DOMAIN">
               <Ambient_Name>UniA</Ambient_Name>
                  <Ambient_Expr Ambient_Type="HOST">
                  <Ambient_Name>Host11</Ambient_Name>
                     <Ambient_Expr Ambient_Type="OBJECT">
                     <Ambient_Name>Lib_App</Ambient_Name>
                     </Ambient_Expr>
                  </Ambient_Expr>
                  <Ambient_Expr Ambient_Type="USER">
                  <Ambient_Name>nmullis</Ambient_Name>
                  </Ambient_Expr>
               </Ambient_Expr>
               ... % other parts of specification
               <Ambient_Expr Ambient_Type="DOMAIN">
               <Ambient_Name>UniB</Ambient_Name>
                  <Ambient_Expr Ambient_Type="HOST">
                  <Ambient_Name>Host21</Ambient_Name>
                     <Ambient_Expr Ambient_Type="OBJECT">
                     <Ambient_Name>Books</Ambient_Name>
                     </Ambient_Expr>
                  </Ambient_Expr>
                  <Ambient_Expr Ambient_Type="USER">
                  <Ambient_Name>mrundell</Ambient_Name>
                  </Ambient_Expr></Ambient_Expr>
            </Composition>
         </Specification>
   </Compilation_Unit>
   <Variables>
      <Identifier name="Library_Web_Server" id_type="HOST"
         id="Host21"/>
      <Identifier name="University_B" id_type="DOMAIN"
         id="UniB"/>
      <Identifier id_type="OBJECT" id="Lib_App"/>
      <Identifier id_type="USER" id="nmullis"/>
   </Variables></Ambient_Calculus_Spec>
```

Figure C.5. Part of the Ambient Calculus Specification for the Library Service.

```
<Constraints>
   <SOD_Constraints>
      <SOD_CR conflicting_set_id="ID_10"
      constraint_id="ID_20" n_conflicting="2"/>
      <SOD_CR conflicting_set_id="ID_11"
      constraint_id="ID_21" n_conflicting="3"/>
      <Conflicting_Sets>
         <Conflicting_Role_Set set_id="ID_10">
            <Conflicting_Role cr_id ="ResAssist"/>
            <Conflicting_Role cr_id ="Lecturer"/>
         </Conflicting_Role_Set>
      <Conflicting_Role_Set set_id="ID_11">
            <Conflicting_Role cr_id ="Librarian"/>
            <Conflicting_Role cr_id ="Sys\_Admin"/>
            <Conflicting_Role cr_id ="Admin"/>
         </Conflicting_Role_Set>
      </Conflicting_Sets>
   </SOD_Constraints>
</Constraints>
```

Figure C.6. Example for SOD constraints specification.

```
<SOD_Conflicts>
   <SOD_Conflict constraint_id="ID_20">
      <Conflicting_User user_id="mrundell">
         <Conflicting_Role role_id="Role1"/>
         <Conflicting_Role role_id="Role2"/>
      </Conflicting_User>
   </SOD_Conflict>
</SOD_Conflicts>
```

Figure C.7. Result of SOD constraints evaluation.

REFERENCES

1. Akar, O., *Model Checking of Ambient Calculus Specifications against Ambient Logic*, M.S. Thesis, Bogazici University, 2009.

2. Grance, T., J. Hash, S. Peck, J. Smith and K. Korow-Diks, *Security Guide for Interconnecting Information Systems*, Technical Report SP 800-47, NIST, 2002.

3. De Capitani di Vimercati, S. and P. Samarati, "Access control in federated systems", *Proceedings of the 1996 workshop on New security paradigms*, ACM, New York, NY, USA, pp. 87–99, 1996.

4. Cardelli, L. and A. D. Gordon, "Ambient Logic", *Mathematical Structures in Computer Science*, 2006.

5. Cardelli, L. and A. D. Gordon, "Mobile ambients", *Theoretical Computer Science*, Vol. 240, No. 1, pp. 177–213, 2000.

6. Burrows, M., M. Abadi and R. Needham, "A logic of authentication", *ACM Transactions on Computer Systems*, Vol. 8, No. 1, pp. 18–36, 1990.

7. Clark, D. D. and D. R. Wilson, "A Comparison of Commercial and Military Computer Security Policies", *Proceedings of the 1987 IEEE Symposium on Research in Security and Privacy*, Oakland, California, USA, IEEE Press, pp. 184–193, 1987.

8. Schuba, C. L., *On The Modeling, Design, And Implementation Of Firewall Technology*, Ph.D. Thesis, Purdue University, 1997.

9. Guttman, J. D., "Filtering Postures: Local Enforcement for Global Policies", *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, Oakland, CA, USA, IEEE Computer Society, Los Alamitos, CA, USA, pp. 120–129, 1997.

10. Fábrega, F. J. T., "Strand spaces: proving security protocols correct", *Journal of Computer Security*, Vol. 7, pp. 191–230, 1999.

11. Lowe, G., "Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR", *Proceedings of the Second International Workshop on Tools and Algorithms for Construction and Analysis of Systems*, Springer-Verlag, London, UK, pp. 147–166, 1996.

12. Paulson, L. C., "Proving Properties of Security Protocols by Induction", *Proceedings of the 1997 IEEE Computer Security Foundations Workshop*, Rockport, Massachusetts, USA, IEEE Computer Society, Los Alamitos, CA, USA, pp. 70–83, 1997.

13. Song, D. X., "Athena: a new efficient automatic checker for security protocol analysis", *Proceedings of the 12th Computer Security Foundations Workshop*, Mordano, Italy, IEEE Computer Society, Los Alamitos, CA, USA, 1999.

14. Kindred, D. and J. M. Wing, "Fast, Automatic Checking of Security Protocols", *Proceedings of the 2nd conference on Proceedings of the Second USENIX Workshop on Electronic Commerce* , Vol. 2, p.5, Oakland, California, USENIX Association, 1996.

15. Kindred, D., *Theory generation for security protocols*, Ph.D. Thesis, Carnegie Mellon University, 1999.

16. Hopper, N. J., S. A. Seshia and J. M. Wing, "A comparison and combination of theory generation and model checking for security protocol analysis", *Proceedings of the Workshop on Formal Methods in Computer Security*, Chigago, Illinois, USA, pp. 100–107, 2000.

17. Becker, M. Y. and P. Sewell, "Cassandra: Flexible trust management, applied to electronic health records", *Proceedings of the 17th IEEE Computer Security Foundations Workshop*, Pacific Grove, CA, USA, IEEE Computer Society, Los Alamitos, CA, USA, pp. 139–154, 2004.

18. Dougherty, D. J., K. Fisler and S. Krishnamurthi, "Specifying and reasoning about dynamic access-control policies", *Proceedings of the Third International Joint Conference on Automated Reasoning*, Seattle, WA, USA, Springer, Berlin, Heidelberg, pp. 632–646, 2006.

19. Unal, D. and M. U. Caglayan, "Theorem proving for Modeling and Conflict Checking of Authorization Policies", *Proceedings of the International Symposium on Computer Networks*, Istanbul, Turkey, IEEE, 2006.

20. Drouineaud, M., M. Bortin, P. Torrini and K. Sohr, "A First Step Towards Formal Verification of Security Policy Properties for RBAC", *Proceedings of the Fourth International Conference on Quality Software*, Braunschweig, Germany, IEEE Computer Society, Los Alamitos, CA, USA, pp. 60–69, 2004.

21. Sohr, K., M. Drouineaud, G. Ahn and M. Gogolla, "Analyzing and Managing Role-Based Access Control Policies", *Knowledge and Data Engineering, IEEE Transactions on*, Vol. 20, No. 7, pp. 924–939, 2008.

22. Becker, M., C. Fournet and A. Gordon, "Design and Semantics of a Decentralized Authorization Language", *Proceedings of the 20th IEEE Computer Security Foundations Symposium*, Venice, Italy, IEEE Computer Society, Los Alamitos, CA, USA, pp. 3–15, 2007.

23. Jajodia, S., P. Samarati, M. L. Sapino and V. S. Subrahmanian, "Flexible sup-

port for multiple access control policies", *ACM Transactions on Database Systems*, Vol. 26, No. 2, pp. 214–260, 2001.

24. Jajodia, S., P. Samarati and V. S. Subrahmanian, "A logical language for expressing authorizations", *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, USA, IEEE Computer Society, Los Alamitos, CA, USA, pp. 31–42, 1997.

25. Bertino, E., F. Buccafurri, E. Ferrari and P. Rullo, "A logical framework for reasoning on data access control policies", *Proceedings of the 12th IEEE Computer Security Foundations Workshop*, Mordano, Italy, IEEE Computer Society, Los Alamitos, CA, USA, pp. 175–189, 1999.

26. Damianou, N., N. Dulay, E. Lupu and M. Sloman, "The Ponder Policy Specification Language", *Proceedings of the International Workshop on Policies for Distributed Systems and Networks*, London, UK, Springer-Verlag, Berlin, Heidelberg, pp. 18–38, 2001.

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

28. Cuppens, F. and C. Saurel, "Specifying a security policy: a case study", *Proceedings of the 9th IEEE Computer Security Foundations Workshop*, Dromquinna Manor, Kenmare, County Kerry, Ireland, IEEE Computer Society, Los Alamitos, CA, USA, pp. 123–134, 1996.

29. Ryutov, T. and C. Neuman, "Representation and Evaluation of Security Policies for Distributed System Services", *Proceedings of DARPA Information Survivability Conference and Exposition*, Hilton Head, SC , USA, IEEE, pp. 172–183, 2000.

30. Milner, R., J. Parrow and D. Walker, "A Calculus of Mobile Processes, I", *Information and Computation*, Vol. 100, No. 1, pp. 1–40, 1992.

31. Milner, R., J. Parrow and D. Walker, "A Calculus of Mobile Processes, II", *Information and Computation*, Vol. 100, No. 1, pp. 41–77, 1992.

32. Cardelli, L. and A. D. Gordon, "Anytime, anywhere: modal logics for mobile ambients", *Proceedings of the 27th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, Boston, Massachusetts, USA, ACM Press, pp. 365–377, 2000

33. Mardare, R. and C. Priami, "A Logical Approach to Security in the Context of Ambient Calculus", *Electronic Notes in Theoretical Computer Science*, Vol. 99, No. 1, pp. 3–29, 2004.

34. Mardare, R., C. Priami, P. Quaglia and A. Vagin, "Model checking biological systems described using ambient calculus", *Proceedings of the 2004 International Conference on Computational Methods in Systems Biology*, Paris, France, Springer, Berlin, Heidelberg, pp. 85–103, 2005.

35. Charatonik, W., S. D. Zilio, A. D. Gordon, S. Mukhopadhyay and J. Talbot, "Model checking mobile ambients", *Theoretical Computer Science*, Vol. 308, No. 3, pp. 277–331, 2003.

36. Scott, D., *Abstracting application-level security policy for ubiquitous computing*, Ph.D. Thesis, University of Cambridge, 2005.

37. Compagnoni, A. and P. Bidinger, "Role-based access control for boxed ambients", *Theoretical Computer Science*, Vol. 398, No. 3, pp. 203–216, 2008.

38. Zhang, N., M. Ryan and D. P. Guelev, "Synthesising verified access control systems through model checking", *Journal of Computer Security*, Vol. 16, No. 1, pp. 1–61, 2008.

39. Braghin, C., N. Sharygina and K. Barone-Adesi, "Automated Verification of Security Policies in Mobile Code", *Proceedings of the 6th International Conference on Integrated Formal Methods*, Oxford, UK, Springer-Verlag, Berlin, Heidelberg, pp. 37–53, 2007.

40. Braghin, C., N. Sharygina and K. Barone-Adesi, "A model checking-based approach for security policy verification of mobile systems", *Formal Aspects of Computing*, Vol. 23, No. 5, pp. 627–648, 2010.

41. Unal, D., O. Akar and M. Ufuk Caglayan, "Model Checking of Location and Mobility Related Security Policy Specifications in Ambient Calculus", In: I. Kotenko and V. Skormin (Editors), *Proceedings of the 5th International Conference on Mathematical Methods, Models and Architectures for Computer Network Security*, St. Petersburg, Russia, Springer-Verlag, Berlin, Heidelberg, pp. 155–168, 2010.

42. Bertino, E., P. Bonatti and E. Ferrari, "TRBAC: A Temporal Role-Based Access Control Model", *ACM Transactions on Information and System Security*, Vol. 4, No. 3, pp. 191–223, 2001.

43. Joshi, J. B. D., E. Bertino, U. Latif and A. Ghafoor, "A Generalized Temporal Role-Based Access Control Model", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 17, No. 1, pp. 4–23, 2005.

44. Hansen, F. and V. Oleschuk, "Spatial Role-Based Access Control Model for Wireless Networks", *Proceedings of the 58th IEEE Vehicular Technology Conference*, Orlando, Florida, USA, IEEE, pp. 2093–2097, 2003.

45. Chandran, S. M. and J. B. D. Joshi, "LoT-RBAC: A Location and Time-Based RBAC Model", *Proceedings of the 6th international conference on Web Information Systems Engineering*, New York, NY, Springer, Berlin, Heidelberg, pp. 361–375, 2005.

46. Ray, I. and M. Toahchoodee, "A Spatio-temporal Role-Based Access Control Model", In: S. Barker and G.-J. Ahn (Editors), *Proceedings of the 21st Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, Redondo Beach, CA, USA, Springer, Berlin, Heidelberg, pp. 211–226, 2007.

47. Kumar, M. and R. E. Newman, "STRBAC - An approach towards spatio-temporal role-based access control", In: S. Rajasekaran (Editor), *Proceedings of the Third IASTED International Conference on Communication, Network, and Information Security*, Cambridge, MA, USA, IASTED/ACTA Press, pp. 150–155, 2006.

48. Damiani, M., E. Bertino, B. Catania and P. Perlasca, "GEO-RBAC: A Spatially Aware RBAC", *ACM Transactions on Information and System Security*, Vol. 10, No. 1, 2007.

49. Kulkarni, D. and A. Tripathi, "Context-aware role-based access control in pervasive computing systems", *Proceedings of the 13th ACM symposium on Access control models and technologies*, Estes Park, CO, USA, ACM, New York, NY, USA, pp. 113–122, 2008.

50. Aich, S., S. Sural and A. K. Majumdar, "STARBAC: spatiotemporal role based access control", *Proceedings of the 2007 OTM confederated international conference on On the move to meaningful internet systems*, Vilamoura, Portugal, Springer-Verlag, Berlin, Heidelberg, pp. 1567–1582, 2007.

51. Aich, S., S. Mondal, S. Sural and A. K. Majumdar, "Role Based Access Control with Spatiotemporal Context for Mobile Applications", *Transactions on Computational Science IV, Lecture Notes in Computer Science*, Vol. 5430, pp. 177–199, Springer-Verlag, Berlin, Heidelberg, 2009.

52. Mondal, S. and S. Sural, "XML-based policy specification framework for spatiotemporal access control", *Proceedings of the 2nd international conference on Security of information and networks*, Sydney, Australia, ACM, New York, NY, USA, pp. 98–103, 2009.

53. Lorch, M., S. Proctor, R. Lepro, D. Kafura and S. Shah, "First experiences using XACML for access control in distributed systems", *Proceedings of the 2003 ACM workshop on XML security*, Fairfax, Virginia, ACM, New York, NY, USA, pp. 25–37, 2003.

54. Sandhu, R. S., E. J. Coyne, H. L. Feinstein and C. E. Youman, "Role-based Access Control Models", *IEEE Computer*, Vol. 29, No. 2, pp. 38–47, 1996.

55. Ferraiolo, D. F., R. Sandhu, S. Gavrila, D. R. Kuhn and R. Chandramouli, "Proposed NIST standard for role-based access control", *ACM Transactions on Information and System Security*, Vol. 4, No. 3, pp. 224–274, 2001.

56. Piromruen, S. and J. Joshi, "An RBAC framework for time constrained secure interoperation in multi-domain environments", *Proceedings of the 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*, Sedona, Arizona, USA, pp. 36 – 45, IEEE Computer Society, Los Alamitos, CA, USA, 2005.

57. Shafiq, B., J. B. Joshi, E. Bertino and A. Ghafoor, "Secure Interoperation in a Multidomain Environment Employing RBAC Policies", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 17, No. 11, pp. 1557–1577, 2005.

58. Bhatti, R., E. Bertino and A. Ghafoor, "X-FEDERATE: A Policy Engineering Framework for Federated Access Management", *IEEE Transactions on Software Engineering*, Vol. 32, No. 5, pp. 330–346, 2006.

59. Bhatti, R., A. Ghafoor, E. Bertino and J. B. D. Joshi, "X-GTRBAC: An XML-Based Policy Specification Framework and Architecture for Enterprise-Wide Access Control", *ACM Transactions on Information and System Security*, Vol. 8, No. 2, pp. 187–227, 2005.

60. Bhatti, R., B. Shafiq, E. Bertino, A. Ghafoor and J. B. D. Joshi, "X-GTRBAC admin: A decentralized administration model for enterprise-wide access control", *ACM Transactions on Information and System Security*, Vol. 8, pp. 388–423, November 2005.

61. Bhatti, R., M. L. Damiani, D. W. Bettis and E. Bertino, "Policy Mapper: Administering Location-Based Access-Control Policies", *IEEE Internet Computing*, Vol. 12, No. 2, pp. 38–45, 2008.

62. Damiani, M. L. and C. Silvestri, "Towards movement-aware access control: Position paper", *Proceedings of the SIGSPATIAL ACM GIS 2008 International Workshop on Security and Privacy in GIS and LBS*, Irvine, California, USA, ACM, New York, NY, USA, pp. 39–45, 2008.

63. "W3C Web Services Glossary", `http://www.w3.org/TR/ws-gloss/`, 2004. accessed at February 2011.

64. Hirschkoff, D., E. Lozes and D. Sangiorgi, "Separability, Expressiveness, and Decidability in the Ambient Logic", *Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science*, Copenhagen, Denmark, IEEE Computer Society, Los Alamitos, California, USA, pp. 423–432, 2002.

65. Hirschkoff, D., E. Lozes and D. Sangiorgi, "On the expressiveness of the Ambient Logic", *Logical Methods in Computer Science*, Vol. 2, No. 2, 2006.

66. Ahn, G.-J. and R. Sandhu, "Role-based Authorization Constraints Specification", *ACM Transactions on Information and System Security.*, Vol. 3, pp. 207–226, 2000.

67. Unal, D. and M. U. Caglayan, "Spatio-Temporal Model Checking of Location and Mobility Related Security Policy Specifications", *Turk. J. Elec. Eng. & Comp. Sci*, accepted, to appear.

68. Giunchiglia, C. C., A. Cimatti, E. Clarke, F. Giunchiglia and M. Roveri, "NUSMV: a new symbolic model checker", *International Journal on Software Tools for Technology Transfer*, Vol. 2, No. 4, pp. 410–425, 2000.

69. Charatonik, W., A. Gordon and J. Talbot, "Finite-Control Mobile Ambients", *Proceedings of the 11th European Symposium on Programming Languages and Systems*, Grenoble, France, Springer-Verlag, London, UK, pp. 295–313, 2002.